INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

CAKE - Hybrides Gruppenschlüsselmanagementprotokoll für RIOT OS

Mehdi Yosofie

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

CAKE - Hybrides Gruppenschlüsselmanagementprotokoll für RIOT OS

Mehdi Yosofie

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Guggemos Dr. Peter Hillmann (Universität Bundeswehr) Klement Streit (Universität Bundeswehr)
Abgabetermin:	23. April 2018

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 23. April 2018

.....
(Unterschrift des Kandidaten)

Vorwort

Diese Bachelorarbeit ist das Ergebnis gemeinschaftlicher Arbeit von Edgar Goetzendorff und Mehdi Yosofie. So ist auch der während dieser Arbeit erstellte Quellcode aus gemeinsamer Mitarbeit entstanden. In dieser Arbeit sind die Kapitel, die jeweils vom anderen Mitwirkenden sind, eindeutig als Zitat markiert. Da die Kapitel 2, 3, 4, 5 und 6 zusammen bearbeitet wurden, steht der Autor jedes Paragraphen jeweils am Rande.

Abstract

Immer mehr Gegenstände werden über das Internet miteinander vernetzt. Sie sind mit Softwaresystemen und Sensoren ausgestattet, wodurch eine Kommunikation zwischen den Objekten ermöglicht wird.
Überall dort, wo mehrere Geräte miteinander in Kontakt stehen, werden oft vertrauliche Informationen ausgetauscht, die Externen nicht zur Verfügung stehen sollen. Demnach benötigen Gruppenkommunikationen eine sichere Übertragung der Daten. Mit geeigneten Verschlüsselungsalgorithmen müssen die ausgehandelten Informationen der Gruppe abgesichert werden. Dadurch soll es Angreifern nicht möglich sein, Gruppenereignisse zu lesen oder zu manipulieren. Es ist ein Kryptosystem notwendig, das diese Anforderungen erfüllt und stets zuverlässig funktioniert. Da eingebetteten Systemen oft nur eingeschränkte Ressourcen vorliegen, spielt neben der reibungslosen Funktionsfähigkeit auch die Zeit und die geringe Netzbelastung eine entscheidende Rolle. Dementsprechend ist es von hoher Bedeutung, dass Anwendungen rasch abgewickelt werden und ihre Datenübertragung so gering wie möglich gehalten wird. Infolgedessen sind Algorithmen erforderlich, die mit starker Effizienz und Effektivität die beschriebenen Herausforderungen lösen.

Ein Gruppenschlüsselmanagementprotokoll wäre ein möglicher Lösungsvorschlag für sichere Kommunikation innerhalb einer Gruppe. CAKE ist ein Gruppenschlüsselverfahren und stellt mit minimalem Rechen- und Nachrichtenaufwand Schlüsselmaterialien für eine Gruppe zur Verfügung. Mit den Gruppenschlüsseln ist den Gruppenteilnehmern eine sichere und effiziente Informationsübertragung möglich. CAKE ist ein hybrides Konzept und integriert Vorteile von verschiedenen Methoden.
Die unterschiedlichen Verfahren wie IKEv2, LKH, SL und GKMP werden in dieser Arbeit erklärt und das daraus entworfene Protokoll vorgestellt. Aufbauend auf dem Protokolldesign wird ein zentraler Server und ein Clientprogramm auf dem Betriebssystem RIOT OS implementiert. Der Server berechnet Gruppenschlüssel und verteilt diese auf effiziente Weise an die Gruppenteilnehmer. Außerdem übernimmt er sämtliche Gruppenoperationen.

Abkürzungsverzeichnis

AI	Authorisierte Instanz
CAKE	Central Authorized Key Extension
CRT	Chinese Remainder Theorem
DH	Diffie-Hellmann
FIFO	First-In-First-Out
GC	Group Controller
GCC	GNU Compiler Collection
GCKS	Group Controller / Key Server
GDOI	Group Domain of Interpretation
GKEK	Group Key Encryption Key
GKMP	Group Key Management Protocol
GKP	Group Key Paket
GMP	GNU Multiple Precision Arithmetic Library
GM	Gruppenmitglied
G-IKEv2	Group Internet Key Exchange version 2
GTEK	Group Traffic Encryption Key
IKE	Internet Key Exchange
IKEv2	Internet Key Exchange version 2
IdD	Internet der Dinge
ISAKMP	Internet Security Association and Key Management Prokoll
KEK	Key Encryption Key
LKH	Logical Key Hierarchy
MCU	Microcontroller Unit
PSK	Pre Shared Key

SA	Security Association
SL	Secure Lock
IPC	Inter Process Communication

Inhaltsverzeichnis

1 Einleitung

Eingebettete Systeme erfordern oft eine geringe Bandbreite und Rechenleistung. Applikationen müssen mit schmalem Softwaredesign ausgestattet sein und mit geringer Übertragungsbandbreite auskommen. In heutigen Systemen sind zahlreiche Geräte miteinander vernetzt, tauschen verschiedenste Informationen miteinander aus und es entstehen riesige Datenmengen. Bei der Übertragung dieser Daten zwischen den Geräten müssen Informationssicherheitsziele wie Vertraulichkeit, Integrität, Authentizität und Verfügbarkeit erfüllt sein.

Diese Ziele sollten auch in gruppenorientierten Kommunikationen realisiert werden, in denen vertrauliche Daten ausgetauscht werden. Deshalb verlangen solche Gruppenkommunikationen ein Kryptosystem, welches Angreifer verhindert, geheime Daten der Gruppe mitzubekommen. Sensible Gruppenereignisse dürfen nur für Gruppenteilnehmer sichtbar und zu entziffern sein.
Im Militärbereich muss die Verständigung zwischen den Soldaten ebenfalls gesichert übertragen werden, sodass feindliche Truppen nicht in der Lage sind, die Militärstrategie abzuhören oder die Kommunikation gar zu manipulieren. Demzufolge sollen nur authorisierte Mitglieder eine Gruppenkommunikation lesen können und nur berechtigte Personen zu einer Gruppe hinzugefügt werden.
Falls spezifiziert wurde, dass ein neuer Teilnehmer einer Gruppe alte Nachrichten nicht lesen darf, so soll er nicht in der Lage sein, mit dem aktuellen Gruppenschlüssel die Nachrichten, die er vor seinem Beitritt empfangen hat, nach seinem Beitritt zu entschlüsseln und zu lesen (*Backward Secrecy*). Diese Nachrichten sollten geheim gehalten werden. Dies erfordert somit eine dementsprechende *Schlüsselunabhängigkeit*. Mit den aktuellen Gruppenschlüsseln in ihrem Besitz dürfen Gruppenteilnehmer nicht an bisherige oder künftige Gruppenschlüssel kommen. Neue Schlüsselmaterialien sollten stets zufällig entstehen. Genauso von Wichtigkeit ist, dass Teilnehmer nach Austritt aus einer Gruppe die im Anschluss empfangenen Nachrichten der Gruppe nicht entschlüsseln können (*Forward Secrecy*). Diese Nachrichten sollen ebenfalls geheimgehalten werden. Die Berechnungen der Gruppeninformationen sollten dabei möglichst effizient und rechenschonend sein, sodass sie in eingeschränkten Umgebungen rasch und zuverlässig verteilt werden. Dementsprechend sollen die Netzbelastung und der Rechenaufwand gering gehalten werden.

Auf Basis von Central Authorized Key Extension (CAKE) entwirft diese Arbeit ein Protokolldesign und implementiert ein minimales Gruppenschlüsselverfahren bestehend aus einer Authorisierten Instanz (AI) und einem Clientprogramm. Die AI - auch Group Controller / Key Server (GCKS) genannt - ist eine vertrauenswürdige Instanz, die die Verwaltung der registrierten Teilnehmer sowie die Berechnungen und Verteilungen der Gruppenschlüssel auf sich nimmt. Das Clientprogramm meldet sich bei der AI an und fordert in verschiedenen Phasen des Protokolls Gruppenschlüssel an.
Die Nachrichtenverteilung vom GCKS zu den Clients wickelt sich je nach Protokollphase über Unicast oder Multicast Nachrichten ab. Dabei soll die Applikation so performant sein,

dass sie die oben beschriebenen Anforderungen erfüllt. Sowohl der Nachrichten- als auch der Berechnungsaufwand sind von großer Bedeutung. Mit CAKE wird dafür in dieser Arbeit eine mögliche Lösung beschrieben.
Der Implementierungsteil dieser Arbeit erfolgt in der Programmiersprache C und als Betriebssystem wird RIOT OS verwendet. RIOT bietet ein eigenes "Network Stack" sowie eine "Crypto" Bibliothek und eignet sich als Programmierumgebung.

Das Ziel dieser Arbeit ist, die Gruppenschlüsselinformationen gemäß CAKE zu berechnen, diese sicher an die Gruppenteilnehmer zu verteilen und somit einen Prototypen eines hybriden Gruppenschlüsselmanagementverfahrens zu implementieren. CAKE basiert auf verschiedene bereits vorhandene Gruppenschlüsselverfahren wie Group Key Management Protocol (GKMP), Secure Lock (SL) und Logical Key Hierarchy (LKH). Die Vorteile dieser Methoden integriert CAKE zu einem gemeinsamen, hybriden Verfahren. Zusätzlich fließen Teile des Internet Key Exchange version 2 (IKEv2) Protokolls in der Registrierungsphase des Clients beim GCKS mit ein, um einen sicheren Kanal zwischen den Parteien zu gewährleisten und einen persönlichen Schlüssel für den Teilnehmer zu generieren.

Weiterer Aufbau dieser Arbeit

Kapitel 2 beschreibt zunächst ein Szenario, woraus die Anforderungen eines Gruppenschlüsselmanagementprotokolls verständlich werden. Darüber hinaus gibt dieser Kapitel einen Überblick über bereits bestehende Gruppenschlüsselprotokolle und Schlüsselmanagementverfahren sowie über das Betriebssystem RIOT OS und warum es für den Einsatz auf kompakten, mobilen Geräten geeignet ist. Die Implementierung dieser Arbeit erfolgt auf RIOT OS. In Kapitel 3 wird das Konzept und die grundlegende Architektur von CAKE erklärt. Zudem werden verschiedene Anforderungen aufgezählt, die das in dieser Arbeit entworfene Protokoll voraussetzt.

Auf den vorherigen Kapiteln aufbauend werden in Kapitel 4 das Protokolldesign sowie die einzelnen Rollen im CAKE System ausführlich beschrieben. Hierbei wird deutlich, aus welchen Komponenten das Protokoll besteht und wie die jeweiligen Algorithmen funktionieren. Anschließend wird in Kapitel 5 auf die Implementierung und die damit verbundenen Datenstrukturen eingegangen. In Kapitel 6 folgt eine Bewertung des konzipierten Protokolls und mit Kapitel 7 endet diese Arbeit mit einer Zusammenfassung und einem Ausblick.

2 Grundlagen

In diesem Kapitel wird in 2.1 zunächst ein mögliches Szenario dargestellt und in 2.2 das ausgewählte Betriebssystem beschrieben. Anschließend werden Gruppenschlüsselprotokolle wie GKMP, Group Domain of Interpretation (GDOI) und Group Internet Key Exchange version 2 (G-IKEv2) vorgestellt und die Schlüsselmanagementverfahren LKH und SL erklärt, auf denen CAKE basiert.

Die vorliegende Arbeit integriert außerdem Teile des G-IKEv2 Protokolls, um einen sicheren Kanal zwischen dem CAKE GCKS und den Clients herzustellen und Unicast Nachrichten sicher zu übertragen. Eine initiale Gruppenerzeugung findet in CAKE mithilfe des broadcastbasierten SL statt. Bei einem Gruppeneintritt (Join Operation) im CAKE System wird bei der Verteilung des neuen Gruppenschlüssels wie beim GKMP vorgegangen. Bei einem Austritt (Leave Operation) wird mit der Idee des LKH und die Berechnungen des SL eine ternäre Baumstruktur verwaltet, damit der Nachrichtenaufwand gering bleibt.

2.1 Szenario

Als möglicher Anwendungsfall für die Verwendung einer gesicherten Gruppenkommunikation kann eine militärische Situation beatrachtet werden, an der die typischen Gruppenoperationen abzuleiten sind (Szenario angepasst aus [12]).

Im Falle eines Einsatzes werden einige Soldaten zu einer Einheit gerufen und bilden gemeinsam eine Gruppe. Zur Unterstützung der Gruppe wird kurz darauf ein weiterer Soldat hinzugesandt. Das entspricht dem Einzeleintritt. Auf dem Weg zum Einsatzgebiet wird die Truppe zudem von einer weiteren Kolonne bekräftigt. Für die weitere Kommunikation müssen beide Gruppen miteinander verschmolzen werden. Am Einsatzgebiet angekommen, wird ein Späher entsandt, während die restlichen Soldaten weiterhin eine geschlossene Einheit bilden. Der Späher wurde hierbei mithilfe einer Einzelaustritt-Operation aus der Gruppe entfernt, und kann somit keine weiteren Gruppen-Nachrichten entschlüsseln. Nach vollendetem Einsatz lösen sich die beiden Teilgruppen und marschieren jeweils zu ihren Lagern. Dies entspricht der Gruppentrennung.

An diesem Szenario werden die verschiedenen Gruppenoperationen ersichtlich, die ein Gruppenschlüsselmanagementprotokoll bereitstellen sollte. Die Soldaten tauschen während der ganzen Kommunikation per Funk ihre Nachrichten aus. Währenddessen darf die Kommunikation durch feindliche Angriffe nicht gelesen werden. Von daher ist eine verschlüsselte Übertragung der Nachrichten enorm wichtig, denn die sensiblen Daten dürfen nur im Kreise der authorisierten Armeeangehörigen bleiben.

2.2 RIOT OS

——— Der folgende Text ist ein direktes Zitat von [7]. ———

n eingebettete Systeme werden eine Vielzahl an Ansprüchen in Bezug auf Zuverlässigkeit, Echtzeitverhalten sowie Leistung gestellt. Ein geeignetes Betriebssystem für diese Art von Systemen muss demzufolge diesen Ansprüchen gerecht werden. Zudem sollte das Betriebssystem in der Lage sein, auf verschiedenster Hardware zu laufen, da es sich bei den eingesetzten Geräten um rechenschwache Microcontroller Units (MCU) bis hin zu stärkeren Geräten mit modernen, effizienten 32-bit Prozessoren handeln kann.

Grundsätzlich kann ein Betriebssystem durch drei Schlüsseleigenschaften klassifiziert werden: (i) Die Struktur des Kernels, (ii) die Funktionsweise des Schedulers, (iii) das Programmiermodell.

(i) Die Struktur des Kernels Der Kernel eines Betriebssystem kann monolitisch oder geschichtet aufgebaut sein oder eine Mikrokernel-Architektur implementieren. Ein monolithischer Kernel ist zwar unkompliziert zu designen, endet aber meist in einer schwer nachvollziehbaren, komplexen Struktur. Durch einen geschichteten Kernel kommt ein hierarchisches Modell hinzu, da als Entwickler entschieden werden muss, wie stark die Trennung zwischen Kernel- und User-Space erfolgt. Bei der Mikrokernel-Architektur wird eine noch höhere Modularität erreicht. Fehler in einzelnen Komponenten bringen nicht das gesamte System zum Absturz; hierdurch wird eine höhere Zuverlässigkeit gewährleistet.

(ii) Funktionsweise des Schedulers Die Wahl der Scheduling-Strategy ist entscheidend für die Unterstützung von Echtzeitverhalten sowie Prioritisierung von Prozessen und Nutzerinteraktionen. Strategien wie First-In-First-Out (FIFO) können beispielsweise den Prozessen keine faire Zuteilung von Resourcen gewährleisten.

(iii) Programmiermodell Ebenfalls von großer Bedeutung ist die Wahl des Programmiermodells. Die Prozesse können entweder alle in ein und dem selben Kontext verarbeitet werden, ohne Segmentierung des Speichers und Zuteilung bestimmter Bereiche des gleichen, oder jeder in seinem eigenen Thread, mit eigenem Speicherstapel.

Im Folgenden werden die zwei vorherrschenden Betriebssysteme für das Internet der Dinge (IdD), Contiki [5] und Tiny OS [15], sowie das vollwertige Betriebssystem Linux verglichen und dargestellt, welche Vorteile RIOT gegenüber diesen bringt.

Tiny OS und Linux implementiere beide einen monolitischen Kernel, während der von Kernel Contiki modular aufgebaut ist, und eher einem geschichteten System entspricht. Das Scheduling von Contiki und Tiny OS erfolgt nach dem FIFO-Prinzip, und ist somit auf die Abarbeitung von kleinen Prozessen optimisiert.

Der Scheduler von Linux kommt der Completely Fair Schedular (CFS) zum Einsatz. Durch die Verwendung eines Rot-Schwarz-Baumes wird eine faire Zuordnung der Prozessorzeit gewährleistet. Das Programmiermodell von Contiki und Tiny OS entspricht einem ereignisbasierten Modell, wobei alle Tasks im selben Kontext abgearbeitet werden. Volles Multi-Threading wird nicht erreicht, jedoch bieten beide Betriebssysteme eine teilweise Multi-Threading Unterstützung.

Die Benutzerfreundlichkeit wird bei Contiki und Tiny OS dadurch beeinträchtigt, das Contiki nur eine Teilmenge der Programmiersprache C zur Verfügung stellt, wodurch bestimmte Schlüsselwörter nicht verwendet werden können, während Tiny OS in einem eigenen C-Dialekt geschrieben wurde (nesC). Linux unterstützt sowohl C als auch C++.

RIOT wurde dahingehend konzipiert, die Unterschiede zwischen klassischen, vollwertigen Betriebssystemen und solchen, die auf eingebettete Systeme ausgerichtet sind, zu überbrücken. Die vorrangigen Ziele des Designs sind eine hohe Energieeffizienz, ein geringer Speicherverbrauch, Modularität, sowie eine einheitliche, von der zugrundeliegenden Hardware unabhängigen, API Schnittstelle.

Der Kernel basiert auf dem FireKernel [21], implementiert eine Mikrokernel-Architektur und ist demzufolge modular. Durch die Modularisierung kann das System auf einen spezifischen Einsatz zugeschnitten werden, wodurch der Speicherverbrauch minimiert wird. Abhängigkeitn zwischen den einzelnen Modulen werden weitestgehend vermieden. Bei dem Scheduler handelt es sich um einen *tickless scheduler*, welcher in den Leerlauf-Modus (*idle thread*) übergeht sobald keine Tasks mehr zur weiteren Verarbeitung zur Verfügung stehen. Dieser idle thread kann nur durch Unterbrechnungssignale verlassen werden. Dadurch, dass die maximale Zeit im idle thread verbracht wird, wird der Stromverbrauch des gesamten Systems minimiert. Durch die Unterstützung der Programmiersprachen C sowie C++, wodurch externe Bibliotheken problemlos eingebunden werden können, sowie der GNU Compiler Collection (GCC) wird eine hohe Benutzerfreundlichkeit für Entwickler erreicht. POSIX Konformität ist ebenfalls (teils) gegeben. RIOT kann für *NIX Systeme kompiliert werden, wodurch die Entwicklung zum Großteil in einer Linux Umgebung stattfinden kann, bevor die Software auf die letztendliche Hardware gespielt wird. Trotz des großen Umfangs an Funktionen, hat RIOT einen sehr kleinen Speicherbedarf. Eine einfache Anwendung benötigt lediglich 5 Kilobyte Festwertspeicher sowie 2 Kilobyte Arbeitsspeicher [1].

In Tabelle 2.1 werden die oben genannten Eigenschaften zusammengefasst und gegenübergestellt.

Tabelle 2.1: Schlüsselaspekte von Contiki, Tiny OS, Linux und Riot

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	○	✗	○	✓	○	○
Tiny OS	<1kB	<4kB	✗	✗	○	✓	✗	✗
Linux	∼1MB	∼1MB	✓	✓	✓	✗	○	○
RIOT	∼1.5kB	∼5kB	✓	✓	✓	✓	✓	✓

(✓) unterstützt, (○) teilweise unterstützt, (✗) nicht unterstützt

RIOT enthält zusätzlich einen vollständigen Network Stack, den Generic (GNRC) Network Stack. Bei dem streng modular aufgebautem GNRC Network Stack wird jedem Modul, demzufolge auch jedem Protokoll, ein eigener Thread zugewiesen. Die Kommunikation unterhalb dieser Threads erfolgt über RIOTs *Inter Process Communication* Modul. Die Struktur dieser Nachrichten ist durch die *netapi* klar definiert. Die einzelnen Module implementieren keine eigenen Warteschlange für Nachrichten (*message queue*, das Inter Process Communication (IPC) Modul allerdings schon. Diese *message queue* ist für die Verwendung des GNRC Network Stack unabdingbar. Einzelnen Threads können über einen entsprechenden Eintrag im Netzwerk Register (*netreg*) Nachrichten eines bestimmten Typs abonnieren [14].

Während eine Nachricht den Network Stack durchläuft, wird sie im Paket Pufferspeicher *pktbuf* hinterlegt.

——— Ende des Zitats. ———

2.3 Gruppenschlüsselmanagementprotokolle

In diesem Kapitel werden verschiedene Gruppenschlüsselprotokolle vorgestellt. Letztendlich verfolgen die in diesem Abschnitt beschriebenen Verfahren GKMP, GDOI und G-IKEv2 dasselbe Ziel: Nämlich Gruppenschlüsselmaterialien gesichert an die Gruppenteilnehmer zukommen zu lassen.

2.3.1 Group Key Management Protocol

Das Group Key Management Protocol (GKMP) ist ein wohlbekanntes Verfahren, das im RFC 2093 und 2094 beschrieben wird. Im GKMP existieren zwei Rollen: der sogenannte Group Controller (GC) und das Gruppenmitglied (GM). Der Group Controller ist ein Gruppenmitglied mit höherer Autorität und übernimmt die Verwaltung kritischer Gruppenaktionen. Er erstellt die Gruppenschlüssel, verteilt diese an die Gruppenmitglieder und übernimmt ebenso weitere Gruppenoperationen wie die des Beitritts-, Austritts- oder Rekey Prozesses. Mit jedem Gruppenmitglied teilt sich der Group Controller jeweils einen geheimen Schlüssel, den Key Encryption Key (KEK).

Dadurch, dass beim GKMP mit jedem Mitglied ein persönlicher Schlüssel ausgehandelt wird, kann bei der Erzeugung einer Gruppe der Group Key Paket (GKP) verschlüsselt mit dem KEK jedes Teilnehmers per Unicast an sie verschickt werden. Der GKP besteht aus einem GTEK und einem GKEK. Der GTEK ist dafür da, um Gruppennachrichten zu ver- und entschlüsseln. Der GKEK existiert, um den GTEK abzusichern.

Tritt ein neues Mitglied in die Gruppe ein, kann der GKP verschlüsselt mit seinem privaten KEK übertragen werden. Die übrigen Clients erhalten das GKP verschlüsselt mit dem bisherigen GKEK. So ist die Backward Secrecy sichergestellt.

Ist ein Rekey notwendig, wird ein neues GKP erstellt und mit dem bisherigen GKEK verschlüsselt und mittels einer Multicast Nachricht an die Gruppenmitglieder verteilt.

Tritt ein Teilnehmer aus einer Gruppe aus, so wird ein neues GKP - bestehend aus einem neuen GTEK und einem neuen GKEK - erstellt, um Forward Secrecy zu gewährleisten. Der neue GKP wird anschließend jeweils mit dem entsprechenden KEK der Teilnehmer, die noch in der Gruppe sind, verschlüsselt und per Unicast an diese versandt. Der Austritt ist mit einem großen Nachrichtenaufwand verbunden, was unvorteilhaft für die Netzkapazität im MANET ist.

In dieser Arbeit wird die Beitrittsoperation wie oben beschrieben durchgeführt. So sind aus Sicht des zentralen Servers zwei Nachrichten an die Gruppenteilnehmer zu verteilen, um Backward Secrecy zu erreichen.

2.3.2 Group Domain of Interpretation

——— Der folgende Text ist ein direktes Zitat von [7]. ———

GDOI ist ein Protokoll zur Verwaltung von Gruppenschlüsseln, bei dem ein GCKS Sicherheitszuweisungen, also kryptographische Richtlinien und Schlüsselmaterial, verteilt. Das Protokoll basiert auf Internet Security Association and Key Management Prokoll (ISAKMP) [17] und Internet Key Exchange (IKE) Version 1 [10] und wird in RFC 6407 [9] spezifiziert. Die in RFC 4046 („Multicast Security (MSEC) Group Key Management Architecture“ [3]) festgelegten Voraussetzungen werden von GDOI erfüllt. ISAKMP definiert zwei Verhandlungsphasen zwischen zwei Einheiten. In der ersten Phase einigen sich beide Einheiten darauf, wie die weitere Kommunikation untereinander gesichert werden soll; hierbei entsteht eine ISAKMP Security Association (SA). In der zweiten Verhandlungsphase werden weitere SAs generiert, die zur Sicherung von weiteren Protokollen genutzt werden können.

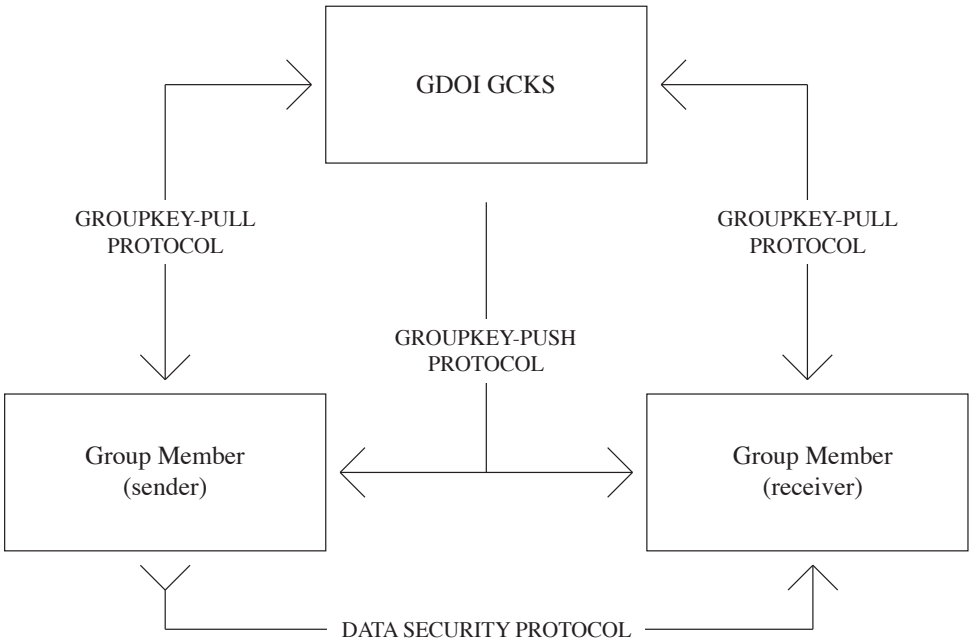

Abbildung 2.1: GDOI Management Model

GDOI erweitert ISAKMP um zwei Protokolle in Phase 2 (Abbildung 2.1):

GROUPKEY-PULL Bei diesem Nachrichtenaustausch wird der Abruf der SAs vom Gruppenmitglied (GM) initiiert. Dieser ist wie oben beschrieben durch ein ISAKMP Phase 1 Protokoll geschützt. Nach dem Austasch ist das Gruppenmitglied, durch den Erhalt der SAs, in der Lage an einer gesichterten Gruppenkommunikation teilzuhaben.

GROUPKEY-PUSH Beim GROUPKEY-PUSH hingegen initiiert der GCKS den Nachrichtenaustausch. Dieser findet in der Regel über Multicast statt und kann somit alle Gruppenmitglieder erreichen. Durch den Austausch können hierbei Mitglieder von der weiteren Gruppenkommunikation ausgeschlossen werden, in dem ihnen durch die versendeten Gruppenrichtlinien die entsprechende Autorisierung entzogen wird.

——— Ende des Zitats. ———

2.3.3 Group Internet Key Exchange version 2

Der G-IKEv2 ist ein Gruppenschlüsselmanagementprotokoll, das sich auf IKEv2 stützt [20]. IKEv2 ist ein IPSec basiertes Protokoll und schafft eine sichere Punkt-zu-Punkt-Übertragung zwischen zwei Kommunikationspartnern.

G-IKEv2 definiert ein ähnliches Gruppenschlüsselprotokoll wie GDOI, wobei dieser IKEv1 verwendet. Ebenso wie GDOI richtet sich G-IKEv2 an die im RFC3740 beschriebenen "Multicast Group (MEC) Security Architecture" und an die "Multicast Security (MSEC) Group Key Management Architecture" im RFC4046.

G-IKEv2 behält die initialen Nachrichtenaustausche des IKEv2 bei und besteht somit initial aus einer IKE_SA_INIT Phase, um einen sicheren Kanal zwischen dem Client und dem vertrauenswürdigen GCKS zu erstellen. Der Initiator fragt am GCKS mit einer Security Association, einem Key Exchange und einer sogenannten Nonce (eine zufällige Zahl) an. Die Security Association umfasst unter anderem, welche Verschlüsselungsalgorithmen verwendet werden sollen. Der Key Exchange beinhaltet den Diffie Hellman Wert des Initiators. Die Antwort vom GCKS beinhaltet dieselben Datenstrukturen mit anderen Werten. Nach dieser Initialisierungsphase werden von beiden Parteien der gemeinsame Schlüssel berechnet, mithilfe dessen die darauffolgenden Nachrichten verschlüsselt übertragen werden.

Anschließend folgt die GSA_AUTH Phase ebenfalls mit zwei Nachrichten, in der sich der Client authentifizieren kann. Hier kann entweder mit einem Pre Shared Key (PSK) oder einem Zertifikat vorgegangen werden, um die Authentifizierung festzustellen.
Diese vier Nachrichtenaustausche in der IKE_SA_INIT und der GSA_AUTH Phase illustrieren die Anmeldung eines Clients beim Server. Anschließend folgen mit dem GSA_REKEY-Austausch Informationen über Schlüsselaktualisierungen. Diese Mitteilungen werden per Multicast nur vom GCKS zu den Clients übertragen und haben keine Erwiderung vom Client zum GCKS zur Folge.

Der Unterschied zwischen IKEv2 und IKEv1 liegt darin, dass IKEv2 in einem einzigem RFC [RFC7296] spezifiziert wird und die RFCs 2407, 2408 und 2409, in denen IKEv1 beschrieben wird, ersetzt. Vor allem aber wurden die initialen Nachrichtenaustausche von acht auf vier reduziert [13], was im Bereich der IdD große Vorteile mit sich bringt.

Einleitend mit einem sicheren Kommunikationskanal, der sich am IKEv2 anlehnt, wurde zusammen mit anderen Verfahren ein hybrides Gruppenschlüsselmanagementprotokoll konzipiert, das neben effizienteren Rekey Berechnungen gleichzeitig wenig Nachrichtenaufwand mit sich bringt. Bei den einzelnen Gruppenoperationen wird dabei stets Forward und Backward Secrecy beachtet.

2.4 Gruppenschlüsselmanagement

——— Der folgende Text ist ein direktes Zitat von [7]. ———

n diesem Kapitel werden zwei Schlüsselmanagementverfahren erläutert. Kapitel 2.4.1 befasst sich mit dem LKH, Kapitel 2.4.2 beschreibt das SL.

LKH ermöglicht einen geringeren Rechenaufwand bei der Schlüsselerzeung, da hierbei meist nur Teilbäume des Schlüsselbaums berücksichtigt werden müssen. Dank dem Secure Lock ist es möglich, den Gruppenschlüssel mit einer einzigen Broadcast Nachricht an die entsprechenden Teilnehmer einer Gruppe zu verteilen. Dabei können unauthorisierte Teilnehmer, die das Datenpaket erhalten, das Secure Lock nicht entschlüsseln, da ihre persönlichen Schlüssel nicht in die Berechnungen des Chinese Remainder Theorem (CRT) eingeflossen sind.

CAKE vereint Eigenschaften aus beiden Protokollen, um die Berechnungen des CRT effizienter zu gestalten und den Nachrichtenaufwand bei Gruppenoperationen zu minimieren.

——— Ende des Zitats. ———

2.4.1 Logical Key Hierarchy

Das LKH Verfahren basiert auf einem Schlüsselbaum (siehe Abbildung 2.2), der vom GCKS verwaltet wird. Jeder Knoten beinhaltet einen KEK. Die Blattknoten repräsentieren die einzelnen Gruppenmitglieder und jeder Blattknoten beinhaltet einen KEK, den entsprechenden persönlichen Schlüssel des jeweiligen Teilnehmers. Jedem Gruppenmitglied werden neben seinem privaten Schlüssel zudem die KEKs aller Knoten, die auf dem Pfad von seinem Knoten bis zur Wurzel liegen, mitgeteilt.

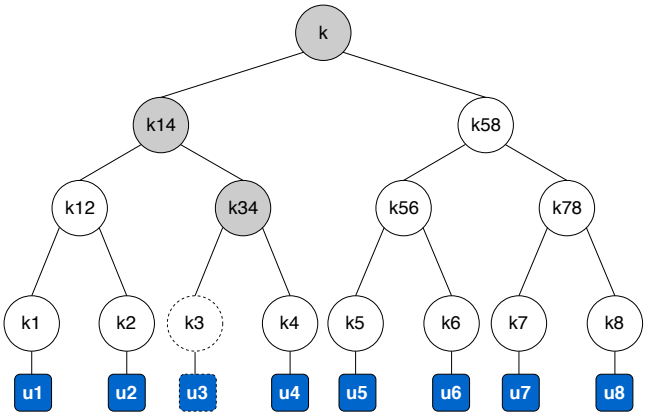

Abbildung 2.2: Der LKH Schlüsselbaum angepasst aus [18]

Betritt ein neuer Teilnehmer u3 die Gruppe, so wird ein neuer Blattknoten im Schlüsselbaum hinzugefügt (in Abbildung 2.2 gepunktet dargestellt). Damit der neue Teilnehmer nicht in Besitz des bisherigen Gruppenschlüssels kommt und somit die Backward Secrecy aufrecht zu erhalten, müssen die betroffenen Knoten k, k14, k34, die auf dem Pfad des neuen Teilnehmers liegen, aktualisiert werden. Es werden neue Schlüssel k', k'14 und k'34 generiert und anschließend werden von der Wurzel aus die Schlüssel mit den jeweiligen Kindknoten verschlüsselt: k wird mit k'14 und k58 verschlüsselt, k'14 wird mit k12 und k'34 verschlüsselt, k'34 wird mit k3 und k4 verschlüsselt. Die Gruppenteilnehmer werden dann über die neuen entsprechenden KEKs informiert. [18]

Der Austritt aus der Gruppe funktioniert ähnlich. Wenn ein Teilnehmer u3 austritt, werden die KEKs aus den Knoten, die auf seinem Pfad bis zur Wurzel liegen, aktualisiert. Im Anschluss werden die neuen Schlüssel k', k'14 und k'34 jeweils mit den entsprechenden

Schlüsseln aus den Kindknoten verschlüsselt. k'34 wird jedoch nur mit k4 verschlüsselt. So kann u3 nicht mehr k'34 entschlüsseln und ist nicht in der Lage, nach oben bis zur Wurzel zu gelangen und den Gruppenschlüssel zu errechnen. Die Forward Secrecy ist demnach sichergestellt.

2.4.2 Secure Lock

——— Der folgende Text ist ein direktes Zitat von [7]. ———

ei dem Secure Lock Verfahren wird zunächst eine Nachricht M mit einem Session Key d verschlüsselt. Der Session Key wird hierbei bei jeder zu versendenen Nachricht neu generiert. Das Secure Lock x wird nun dazu verwendet, den *session key* zu verschlüsseln. Die verschlüsselte Nachricht wird im Anschluss an den verschlüsselten Session Key angehängt, woraus sich der folgende in Abbildung 2.3 dargestellte Aufbau ergibt.

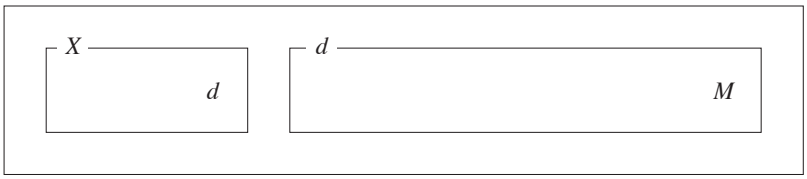

Abbildung 2.3: Aufbau einer mit dem Secure Lock gesicherten Nachricht

Das Secure Lock x muss dabei die folgenden Eigenschaften erfüllen:

- Es darf nur von denjenigen Nutzern geöffnet werden können, für die die Nachricht bestimmt ist
- Es muss vom Session Key d abhängig sein

Um beide dieser Eigenschaften zu erfüllen, muss eine funktionelle Abhängigkeit zwischen dem Secure Lock und dem eigentlichen Schlüssel, als auch zwischen dem Secure Lock und dem Session key bestehen. Im weiteren Verlauf dieses Kapitels wird erläutert, wie ein solches Secure Lock unter Verwendung des Chinesichen Restsatzes (im Folgenden *CRT*) konstruiert werden kann.

Seien $N_1, N_2, \ldots, N_n$ natürliche, paarweise teilerfremde Zahlen, $R_1, R_2, \ldots, R_n$ natürliche Zahlen und $L = N_1 * N_2 * \ldots * N_n$, dann gibt es ein System von Kongruenzen

$$\begin{aligned} X &\equiv R_1 \; mod \; N_1 \\ X &\equiv R_i \; mod \; N_i \\ X &\equiv R_n \; mod \; N_n \end{aligned}$$

mit einer gemeinsamen Lösung X im Intervall $[1, L-1]$ und

$$\begin{aligned} x &= \left(\sum_{i=1}^{n} (L/N_i) * R_i * f_i \right) \; mod \; L \\ \text{mit} \quad & 1 \equiv f_i * (L/N_i) \; mod \; N_i), \; R_i \leq N_i. \end{aligned}$$

Beim Secure Lock ist nun $R_i = E_{ek_i}(d)$, wobei E die Verschlüsselungsoperation und ek_i den privaten Schlüssel eines Nutzers u_i notiert.

$$X \equiv (E_{ek_1}(d)) \ mod\, N_1$$
$$X \equiv (E_{ek_i}(d)) \ mod\, N_i$$
$$X \equiv (E_{ek_n}(d)) \ mod\, N_n$$

Ein Nutzer u_i, dessen privater Schlüssel ek_i und natürliche Zahl N_i mit in die Berechnung eingeflossen ist, kann über die Umkehrfunktion $X \ mod\, N_i$ seinen Wert R_i berechnen, aus welchem er wiederum den Session Key d ableiten, und letztendlich die Nachricht M entschlüsseln kann [4].

——— Ende des Zitats. ———

3 Central Authorized Key Extension - CAKE

CAKE ist ein hybrides Gruppenschlüsselmanagementverfahren, das für eine sichere Kommunikation innerhalb einer Gruppe sorgt, wobei strikte Sicherheitsanforderungen und geringe Netzbelastung im Vordergrund stehen. CAKE erfüllt diese Voraussetzungen mit minimalem Kommunikations- und Berechnungsaufwand. Die Schlüsselverwaltung ist zentral vom CAKE GCKS organisiert. Der CAKE GCKS führt die Schlüsselberechnungen durch und verteilt die Schlüssel an die entsprechenden Mitglieder einer spezifizierten Gruppe. Im Abschnitt 3.1 folgen weitere Erklärungen zum CAKE System und in Abschnitt 3.2 werden Anforderungen aufgezählt, die CAKE benötigt.

3.1 Definition und Erklärung

Nur authorisierte Teilnehmer können im CAKE System einer gesicherten Gruppenkommunikation beitreten und Gruppenschlüsselinformationen erhalten. Dazu müssen sie sich dementsprechend authentifizieren. In dieser Arbeit geschieht dies durch einen PSK. Hinzu kommt, dass jeder neue Teilnehmer, der die Registrierungsphase durchläuft, zwei geheime Schlüssel zugewiesen bekommt, einen KEK_i und eine Primzahl m_i. Mit dem persönlichen KEK werden Nachrichten vom GCKS zum Client abgesichert. Zudem fließen beide Schlüssel in die Berechnungen des CRT ein.
Das Zustandekommen des persönlichen KEK jedes Teilnehmers schreibt CAKE nicht vor. In dieser Arbeit beruht die Entstehung des KEK wie im auf das Diffie-Hellmann (DH) basierende IKEv2 Protokoll. Der persönliche Schlüssel könnte ebenso auf das Verfahren der RSA beruhen. Weiterhin kann die Authentifizierung des Clients mit dem vertrauenswürdigen GCKS statt einem PSK via Zertifikat erfolgen.
Im CAKE System existiert neben dem geheimen Schlüsselpaar jedes Teilnehmers (KEK_i, m_i) das Gruppenschlüsselpaar aus Group Traffic Encryption Key (GTEK) und Group Key Encryption Key (GKEK). Der GTEK wird zur Verschlüsselung von Gruppennachrichten benutzt und für die eigentliche Gruppenkommunikation verwendet. Jeder Teilnehmer einer Gruppe muss diesen Schlüssel besitzen. Der GKEK ist dafür da, den GTEK abzusichern. Dazu wird der GTEK mit dem GKEK bitweise mithilfe der XOR Operation verschlüsselt. Gemäß dem One-Time-Pad bietet die Verwendung der XOR Funktion informationstheoretische Sicherheit [12]. Der One-Time-Prinzip ist die stärkstmögliche Verschlüsselungstechnik und der damit chiffrierte Geheimtext gilt als “unbrechbar” (engl. holocryptic)[6] [2]. So wird der eigentliche Gruppenschlüssel GTEK zusätzlich abgesichert.

Unterteilung in GTEK und GKEK

Ein separater GKEK zur Verbreitung eines neuen GTEK ist notwendig, da die Nutzung des aktuellen $\text{GTEK}_{\text{aktuell}}$ als $\text{GKEK}_{\text{aktuell}}$ bei bitweisen XOR dazu führt, dass der neue Teilnehmer die alten, mit dem $\text{GTEK}_{\text{aktuell}}$ gesicherten Nachrichten entschlüsseln könnte. Durch Anwendung von XOR auf die abgefangene Nachricht mit GTEK_{neu}

ergibt sich $GTEK_{aktuell}$. Dies wird durch den Einsatz eines separaten GKEK verhindert.[12]

3.2 Anforderungen von CAKE

——— Der folgende Text ist ein direktes Zitat von [7]. ———

as in dieser Arbeit präsentierte Protokoll stellt verschiedene Anforderungen an das zugrunde liegende System. Diese Anforderungen können in Sicherheitsanforderungen, funktionale und nicht-funktionale Anforderungen unterteilt werden.

——— Ende des Zitats. ———

3.2.1 Sicherheitsanforderungen

Zu den sicherzustellenden Schutzzielen und Anforderungen zählen Forward und Backward Secrecy, Schlüsselunabhängigkeit sowie Kollisionsfreiheit. Zusätzlich sollten weitere Informationssicherheitsziele wie Authentizität, Vertraulichkeit und Integrität eingehalten werden [19]. Diese werden im Folgenden jeweils knapp zusammengefasst.

Forward Secrecy

Verlässt ein Teilnehmer eine gesicherte Gruppenkommunikation, so darf und soll er nicht in der Lage sein, die nach seinem Austritt empfangenen Nachrichten zu entschlüsseln. Diese ausgetauschten Nachrichten sollen geheimgehalten werden und nur für aktuelle Gruppenmitglieder zu entschlüsseln sein.

Backward Secrecy

Tritt ein Teilnehmer in eine bestehende Gruppe ein, so darf und soll er nicht in der Lage sein, die vor seinem Beitritt empfangenen Nachrichten zu entschlüsseln. Diese Nachrichten sollen ebenfalls geheimgehalten werden.

Schlüsselunabhängigkeit / Folgenlosigkeit

Durch den Besitz eines Schlüssels darf es nicht möglich sein, auf andere Schlüssel zu schließen.

Kollisionsfreiheit

Begeht eine Teilmenge einer Gruppe ein unerlaubtes Vorgehen, so führt das nicht zum Schaden eines einzelnen Gruppenmitgliedes.

Authentizität

——— Der folgende Text ist ein direktes Zitat von [7]. ———

as Protokoll muss Verfahren umfassen, die es sowohl den Clients als auch der Server erlauben, sich gegenseitig Vertrauenswürdigkeit und Echtheit ihrer Nachrichten zu beweisen.

——— Ende des Zitats. ———

Vertraulichkeit auf Gruppenebene

Gruppenschlüsselinformationen, die für eine Gruppe spezifiziert sind und weitere Daten, die innerhalb der Gruppe ausgestauscht werden, dürfen von Externen nicht gelesen werden können.

Datenintegrität

Sicherstellung, dass Daten beim Übertragen nicht verändert werden.

3.2.2 Funktionale Anforderungen

——— Der folgende Text ist ein direktes Zitat von [7]. ———

unktionale Anforderungen beschreiben die Funktionaliäten, die eine Implementierung zur Verfügung stellen muss. Das System um CAKE muss in der Lage sein, Nachrichten zu versenden und zu empfangen. Es muss Funktionen zur Erstellung, Löschung und Verwaltung von Gruppen ermöglichen. Durch die Verwendung von RIOT und dem dazugehörigen GNRC Network Stack ist diese Anforderung erfüllt. Diese Nachrichten müssen außerdem mittels kryptographischer Verfahren verschlüsselt und entschlüsselt werden können. Ebenfalls muss es in der Lage sein das CRT zu berechnen und aufzulösen, sowie geeignetes Schlüsselmaterial zu erzeugen. Im Folgenden werden weitere Anforderungen prägnant aufgezählt, die ein Gruppenschlüsselprotokoll benötigt.

——— Ende des Zitats. ———

Multicast Kommunikation

Der GCKS sollte neben Unicast Nachrichten ebenso multicastbasierte übertragen können, um Gruppenfunktionen zu verwalten und Schlüsselinformationen zu verschicken.

Gruppenoperationen

Ein modernes Gruppenschlüsselmanagementprotokoll beinhaltet viele Gruppenfunktionen. Durch das beschriebene Szenario ergeben sich folgende Anforderungen an die Gruppenoperationen. Dabei müssen weiterhin die oben beschriebenen Sicherheitsanforderungen bestehen bleiben.

- **Einzel-Eintritt**
  Ein Teilnehmer fragt beim Gruppenmanager an, in eine bestehende Gruppe einzutreten und wird von diesem hinzugefügt. Dabei muss die Backward Secrecy gewährleistet sein.
- **Mehrfach-Eintritt**
  Mehrere Teilnehmer fragen beim Gruppenmanager an, in eine bestehende Gruppe einzutreten und werden von diesem hinzugefügt. Dabei muss die Backward Secrecy gewährleistet sein.
- **Einzel-Austritt**
  Ein Gruppenmitglied fragt beim Gruppenmanager an, aus einer Gruppe auszutreten. Dabei muss die Forward Secrecy beachtet werden.

- **Mehrfach-Austritt**
  Mehrere Gruppenmitglieder fragen beim Gruppenmanager an, aus der Gruppe auszutreten. Dabei muss die Forward Secrecy beachtet werden.
- **Einzel-Entfernung / Einzel-Suspendierung**
  Ein Gruppenmitglied kann vom Gruppenmanager aus einer Gruppe entfernt werden, ohne dass der Teilnehmer selbst eine Austrittsanfrage stellt. Die Forward Secrecy muss sichergestellt sein.
- **Mehrfach-Entfernung / Einzel-Suspendierung**
  Mehrere Gruppenmitglieder können vom Gruppenmanager aus einer gesicherten Gruppenkommunikation entfernt werden, ohne dass die Teilnehmer selbst eine Austrittsanfrage stellen. Die Forward Secrecy muss dabei eingehalten werden.
- **Einzel-Zwangseintritt/ Einzel-Einladung**
  Ein Gruppenmitglied kann vom Gruppenmanager in eine Gruppe hinzugefügt werden, ohne dass er selbst eine Beitrittsanfrage stellt. Die Backward Secrecy muss beachtet werden.
- **Mehrfach-Zwangseintritt/ Mehrfach-Einladung**
  Mehrere Gruppenmitglieder können vom Gruppenmanager in eine gesicherte Gruppenkommunikation hinzugefügt werden, ohne dass sie selbst eine Beitrittsanfrage stellen. Die Backward Secrecy muss dabei eingehalten werden.
- **Re-Keying**
  Die Aktualisierung des Gruppenschlüssels muss über eine effiziente Vorgehensweise möglich sein. Gleichzeitig darf die Forward und Backward Secrecy nicht gefährdet sein.
- **Gruppenverschmelzung**
  Mehreren Gruppen ist durch Re-Keying ein gemeinsamer Schlüssel effizient bereitzustellen. Dabei muss die Backward Secrecy gewährleistet sein.
- **Gruppenteilung**
  Eine Gruppe teilt sich in mehrere Teilgruppen auf. Die Forward Secrecy muss hier ebenfalls sichergestellt sein.

Alle oben beschriebenen Gruppenoperationen gelten nur für authorisierte und im CAKE System registrierte Teilnehmer.

Kryptographische Anforderungen

——— Der folgende Text ist ein direktes Zitat von [7]. ———

ie Verschlüsselung von Nachrichten ist für das Protokoll unabdingbar, da es vor allem für Funktechnologien konzipiert wurde, wobei Nachrichten leicht abgefangen werden können. Dies setzt bestimmte Bedingungen voraus. Diese beinhalten die Benutzung von mindestens einem Verschlüsselungsalgorithmus, einem Algorithmus für Integritätswahrung, einen Generator für Zufallszahlen ausreichender Güte und eine Diffie-Hellman Gruppe für den Aufbau eines sicheren Kanals zwischen Client und Server.

——— Ende des Zitats. ———

3.2.3 Nicht-funktionale Anforderungen

——— Der folgende Text ist ein direktes Zitat von [7]. ———

as Protokoll muss einer Reihe von nicht-funktionalen Anforderungen genügen. Diese sind in Rechenleistung, Speicherkapazität und Eneergieffizienz gegliedert.

Zuverlässigkeit

Das System muss eventuell auftretende Fehler abfangen, verarbeiten und in der Lage sein, sich entsprechend wieder zu erholen.

Skalierbarkeit

Das Protokoll muss in der Lage sein, eine größere Anzahl an Clients pro Gruppe zu unterstützen. Die zu unterstützende Anzahl wird auf 81 Teilnehmer begrenzt.

Rechenleistung

Da die eingesetzten Geräte über eine geringe Rechenleistung verfügen, muss der benötigte Aufwand für die Verschlüsselung, Entschlüsselung sowie die Erzeugung von Schlüsselmaterial, insbesondere durch das CRT, möglichst gering gehalten werden.

Speicherkapazität

Der verfügbare Arbeitsspeicher auf den Geräten muss ausreichend groß sein, um das Betriebssystem RIOT laden zu können. Hinzu kommt der für das Schlüsselmaterial, die Gruppen, und die Clients benötigte Speicher. Der Client sollte über ausreichend Arbeitsspeicher verfügen, um mindestens einer Gruppe beitreten zu können.

Energieeffizient

Das Protokoll ist für kompakte, portable Geräte ohne konstante Stromversorgung konzipiert. Als solches muss die Anzahl und der Aufwand der benötigten Rechenoperationen, damit auch der Stromverbrauch, möglichst gering gehalten werden.

——— Ende des Zitats. ———

4 Design

——— Der folgende Text ist ein direktes Zitat von [7]. ———

ieses Kapitel dient der Beschreibung der Architektur und des Designs des Protokolls, in das CAKE eingegliedert wurde, unter Einbezug der in Kapitel 3.2 definierten Anforderungen.

Das Protokoll benutzt den UDP Port 849, da dieser zum Entstehungszeitpunkt dieser Arbeit noch nicht von der *Service Name and Transport Protocol Port Number Registry* vergeben wurde und somit zur freien Verfügung steht. Das G-IKEv2 Protokoll [11], an das sich dieses Protokoll teilweise anlehnt, benutzt UDP Port 848.

Nach der initialen Registrierung eines Clients, ist dieser in der Lage die Erstellung einer neuen Gruppe anzufordern, Gruppen beizutreten sowie Nachrichten zu verschicken. Die Registrierung, sowie die für die Gruppenverwaltung notwendigen Nachrichten werden via Unicast übermittelt. Der Client erwartet auf jede von ihm an den Server übermittelten Unicast-Nachricht eine Antwort. Diese enthält, im Falle einer erfolgreich ausgeführt Aktion, die angeforderten Daten, oder, im Falle eines Verarbeitungsfehlers einen entsprechenden Fehler-Code. Nachrichten innerhalb einer Gruppe, sowie einige der für die Schlüsselverteilung notwendigen Nachrichten werden mittels Multicast übermittelt und müssen nicht quittiert werden.

Kapitel 4.1 beschreibt die verschiedenen Rollen innerhalb des Protokolls, Kapitel 4.2 beschreibt den Aufbau eines sicheren Kanals zwischen einem Client und dem GCKS, in Kapitel 4.3 werden die verschiedenen Gruppenoperationen erläutert.

——— Ende des Zitats. ———

4.1 Rollenmodell

——— Der folgende Text ist ein direktes Zitat von [7]. ———

us dem in Kapitel 2 beschriebenen Szenario und den definierten Anforderungen lassen sich vier verschiedene Rollen ableiten: (i) Key Server, (ii) Group Controller, (iii) Client und (iv) Group Manager.

(i) Key Server Die wesentliche Aufgabe des Servers besteht darin, das Schlüsselmaterial sowie die Schlüsselbäume zu verwalten. Er ist für die Berechnung und Erneuerung des Schlüsselmaterials verantworlich.

(ii) Group Controller Der Group Controller hat die Aufgabe alle Clients und Gruppen zu verwalten. Bei Bedarf fordert er den Key Server dazu auf, neues Schlüsselmaterial zu berechnen, welches er an die Clients verteilt.

(iii) Client Der Client entspricht dem tatsächlichen Gruppenmitglied. Als solcher kann er Schlüsselmaterial empfangen, sowie Nachrichten empfangen und versenden.

(iv) Group Manager Für die organisatorischen Bedürfnisse innerhalb einer Gruppe ist der Group Manager zuständig. Abhängig von der Implementierung ist dieser nicht zwingend selber ein Mitglied der Gruppe, die er verwaltet. Der Group Manager kann, zusätzlich zu den Berechtigungen eines normalen Clients, Clients aus der Gruppe ausschließen, die Löschung seiner Gruppe beantragen sowie den Key Server dazu anstoßen, einen Rekey-Prozess einzuleiten.

Der Nachrichtenaustausch von der Registrierung eines Clients bis hin zur ersten Gruppenoperation wird in Abbildung 4.1 veranschaulicht.

——— Ende des Zitats. ———

4.2 Aufbau eines sicheren Kanals

——— Der folgende Text ist ein direktes Zitat von [7]. ———

er Aufbau des sicheren Kanals zwischen dem GCKS und einem Client ist stark an das G-IKEv2-Protokoll angelehnt und besteht aus einem CAKE_INIT sowie einem CAKE_AUTH Nachrichtenaustausch. Die einzelnen Payloads dieser Nachrichten werden in Kapitel 4.2.1 und 4.2.2 beschrieben und in Abbildung 4.2 dargestellt.

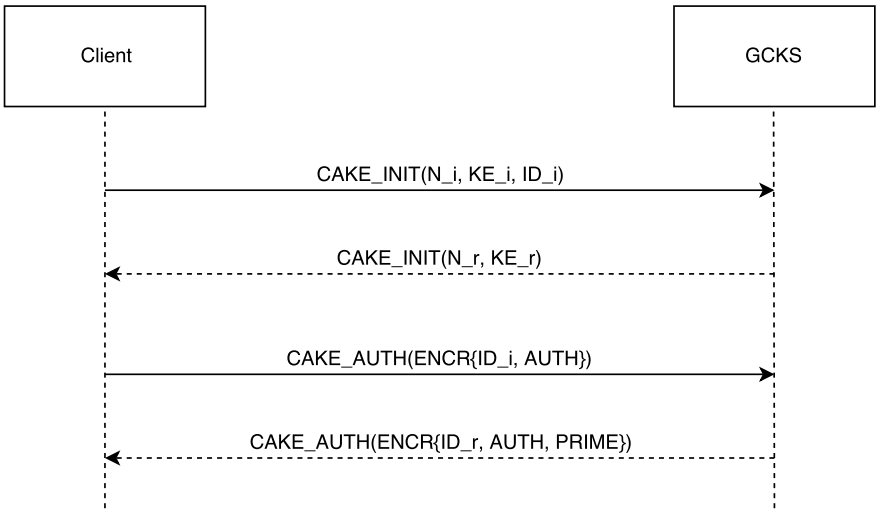

Abbildung 4.2: Aufbau eines sicheren Kanals

——— Ende des Zitats. ———

4.2.1 CAKE_INIT

——— Der folgende Text ist ein direktes Zitat von [7]. ———

er ersten Nachrichten des Protokolls sind die zwei CAKE_INIT Nachrichten, die sich der registrierende Client und der GCKS zuschicken. Der Nachrichtenaustausch wird hierbei stets

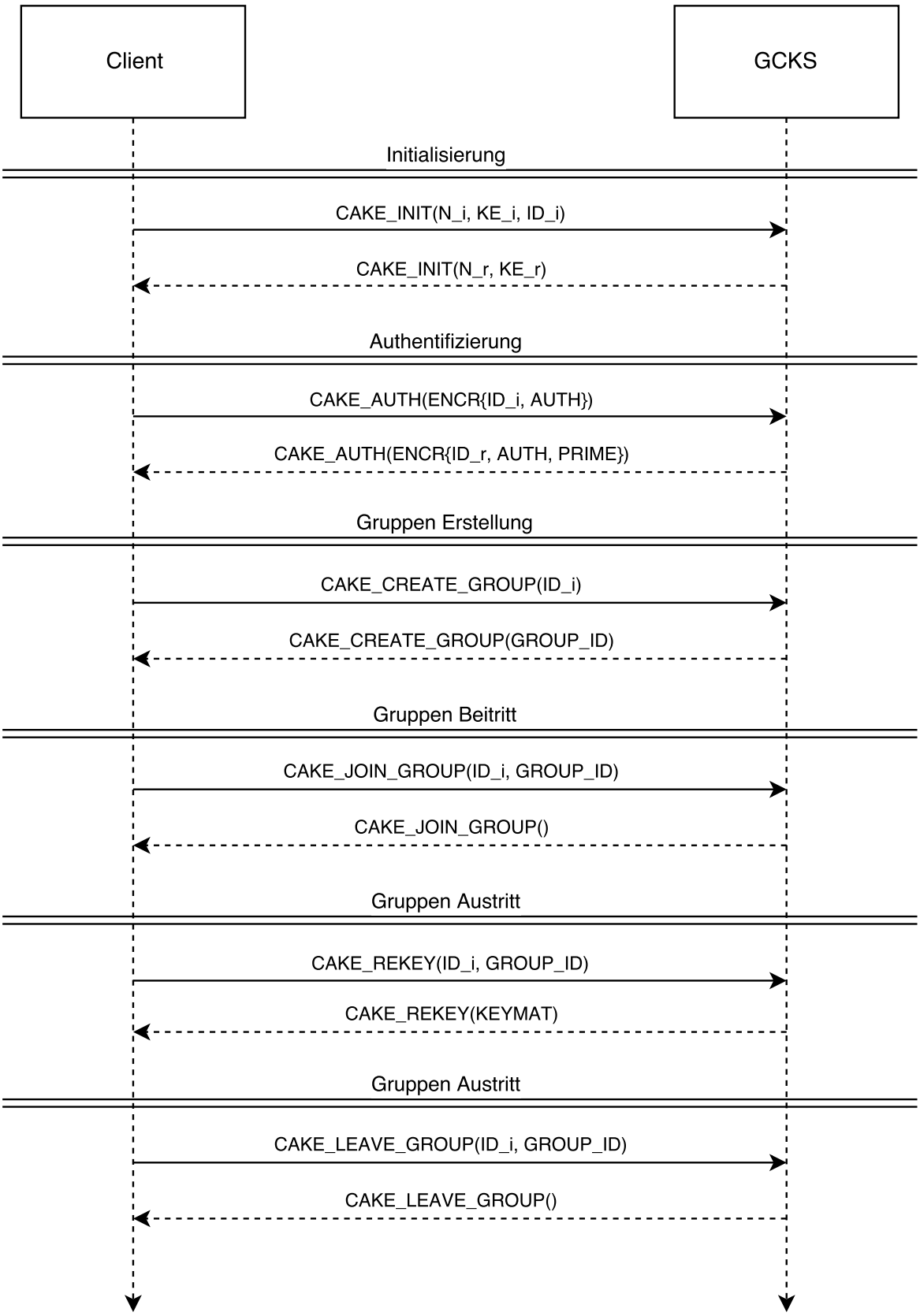

Abbildung 4.1: Sequenzdiagram

vom Client initiiert. Der Client schickt dem GCKS die folgenden Payloads: eine Zufallszahl (Nonce) N_i und den Wert für den Diffie-Hellman-Schlüsselaustausch KE_i. Die kryptographischen Algorithmen werden derzeit nicht zwischen Client und GCKS ausgehandelt, sondern in der entsprechenden Konfigurations-Datei definiert. Der GCKS antwortet auf diese Nachricht mit seiner Nonce N_r und seinem Diffie-Hellman-Wert KE_r. Beide Parteien sind nun in der Lage, ein gemeinsames Geheimnis zu berechnen, aus welchem letztendlich das Schlüsselmaterial für die Verschlüsselung der folgenden Kommunikation abgeleitet wird.

——— Ende des Zitats. ———

4.2.2 CAKE_AUTH

——— Der folgende Text ist ein direktes Zitat von [7]. ———

m sich zu authentifizieren sendet der Client den Identitäts-Payload ID_i sowie einen Authentifizierungs-Payload $AUTH$, der das für die Authentifizierung benötigte Geheimnis, den Pre Shared Key (PSK), enthält. Diese Payloads werden mit dem zuvor abgeleiteten Schlüsselmaterial gesichert. Der PSK wird vom GCKS überprüft, welcher bei einer Übereinstimmung seinerseits dem Client den Identitäts-Payload ID_r, den Authentifizierungs-Payload $AUTH$ sowie eine Primzahl schickt. Hierbei wird für jede Kommunikationsrichtung ein eigener Schlüssel verwendet.

——— Ende des Zitats. ———

4.3 Gruppenoperationen

Im beschriebenen Szenario ist es von großer Bedeutung, eine Gruppe sicher und rasch zu erstellen. Hinzu kommt, dass Teilnehmer sowohl relativ unkompliziert in eine Gruppenkommunikation hinzugefügt, als auch entfernt werden sollen. Diese Operationen sind grundlegend wichtig für ein Gruppenschlüsselprotokoll und dürfen in diesem Kontext nicht fehlen.
In diesem Kapitel werden diese Aspekte des CAKE Protokolls ausführlich erklärt. Im Abschnitt 4.3.1 wird beschrieben, wie der GCKS im CAKE System eine Gruppe erzeugt. Der Einzeleintritt und der Einzelaustritt werden im Abschnitt 4.3.2 bzw. 4.3.3 erläutert. In 4.3.4 werden die Operationen Mehrfacheintritt und Mehrfachaustritt formuliert und die Aktualisierung des Gruppenschlüssels wird unter “Rekey einer Gruppe” 4.3.5 beschrieben.

4.3.1 Initiale Erzeugung einer Gruppe

In CAKE sind die grundlegenden Informationen einer Gruppe der GTEK und der GKEK. Wofür und warum sie beide benötigt werden, wurde bereits in 3.1 ausgeführt. Dieses Schlüsselpaar muss gesichert an die entsprechenden Teilnehmer einer spezifizierten Gruppe verteilt werden, die vorher bereits im CAKE System registriert sein müssen. Wie sich ein Client in diesem hybriden Gruppenschlüsselprotokoll anmeldet, wurde im Abschnitt 4.2.1 und 4.2.2 beschrieben.

Um eine Gruppe initial zu erzeugen, erstellt der GCKS zufällig einen GTEK und einen GKEK. Für die Verteilung des Schlüsselpaares wird ein CRT berechnet. Die Primzahlen aller Teilnehmer einer Gruppe fließen in die Berechnungen des CRT ein. Als zweiter Input für den

CRT fließt der GKEK verschlüsselt mit dem persönlichen Schlüssel KEK_i jedes Teilnehmers mit ein.

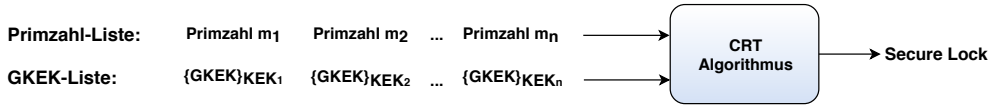

Abbildung 4.3: Entstehung des Secure Lock (SL)

Entscheidend für den Rechenvorgang des CRT auf Seiten des Servers und dem Auflösen des CRT seitens des Clients ist, dass die Primzahlen der Teilnehmer in der selben Reihenfolge wie der jeweils mit dem KEK jedes Teilnehmers veschlüsselte GKEK vorliegen. Das bedeutet, die Primzahlen m_1 bis m_n, die jeweils einem Client zugeordnet sind, müssen sich in der gleichen Reihenfolge befinden wie die Verschlüsselung des GKEKs mit dem KEK_i jedes Teilnehmers (Verdeutlichung in Abbildung 4.3). Im Algorithmus des CRT folgen Multiplikationen und Modulo Berechungen mittels der Primzahl- und GKEK-Listen. Danach ist der Output des CRT, der *Secure Lock* (SL), abgeschlossen. Nur so kann das broadcastbasierte Datagramm auf Seiten des Clients aufgelöst werden und zum GKEK führen, denn allein er kann mit seiner Primzahl nur das mit seinem KEK verschlüsselte GKEK erhalten. Der Secure Lock bildet den ersten Teil des zu versendenden Datagramms (siehe die nachfolgende Abbildung 4.4). Der GTEK wird bitweise mittels XOR mit dem GKEK verschlüsselt und vervollständigt den zweiten Teil der Dateneinheit. Das Paket wird im Header mit NEW_GROUP versehen und per Multicast an alle Mitglieder der spezifizierten Gruppe versandt.

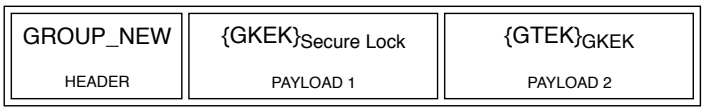

Abbildung 4.4: Datagramm bei der initalen Erzeugung einer Gruppe

Ein Client kann das Secure Lock nur dann auflösen, wenn seine geheime Primzahl in der Berechnung des CRT mit enthalten ist. Nachdem ein Gruppenmitglied die Modulo Funktion mithilfe seiner Primzahl auf das empfangene Secure Lock angewandt und so den entsprechenden verschlüsselten GKEK erhalten hat, kann er den GKEK mit seinem privaten Schlüssel entschlüsseln und den GKEK abspeichern. Mithilfe des GKEK bekommt er nun den GTEK, indem die XOR Funktion auf den überschlüsselten GTEK, der sich im zweiten Teil des Datenpaketes befindet, durchgeführt wird.
Unauthorisierte Mitglieder können nicht zum Gruppenschlüssel gelangen, da ihre Primzahl nicht in der CRT Rechnung miteinbezogen wurde.

Sind die Teilnehmer im CAKE System angemeldet und wurden sie für eine gesicherte Gruppenkommunikation ausgewählt, so ist nur eine einzige Multicast Nachricht notwendig, um allen Teilnehmern einer spezifizierten Gruppe über den GTEK zu informieren. Bei der initialen Erzeugung einer Gruppe besteht zwar aufgrund der Berechnungen des CRT ein erhöhter Rechenaufwand für den GCKS, jedoch beträgt dafür die Nachrichtenzahl eins. Dies ist im beschriebenen Szenario perfekt geeignet, da Kommunikationsübertragung über Funk teuer ist, denn die Netzbelastung im MANET sollte gering gehalten werden.

4.3.2 Eintritt von neuen Teilnehmern in eine Gruppe - Join

Wird ein Client in eine gesicherte Gruppenkommunikation hinzugefügt, muss er zunächst die Anmeldungsphasen von CAKE durchlaufen. Hierzu handelt er mit dem GCKS sein geheimes Schlüsselpaar aus: Seinen persönlichen Schlüssel KEK_i und seine ebenfalls geheime Primzahl m_i. Beide Schlüssel sind nur dem GCKS und dem entsprechenden Client bekannt und ebenfalls beide Schlüssel fließen in die Berechnungen des CRT ein. Nachdem sich der Client beim CAKE-Server registriert und beim GCKS eine Betrittssanfrage für eine Gruppe gemacht hat, beginnt der Join Algorithmus. Hierbei wird aber gegenüber der initialen Erzeugung der Gruppe kein CRT berechnet.

Dazu erstellt der GCKS ein neues Gruppenschlüsselpaar GTEK_{neu} und GKEK_{neu}. Der Gruppenschlüssel muss zwingend aktualisiert werden, da der neue Teilnehmer die mit dem $GTEK_{aktuell}$ verschlüsselten Nachrichten, die er eventuell im Vornherein empfangen hat, lesen könnte. Durch die Aktualisierung des Gruppenschlüssels werden somit die alten Nachrichten geheimgehalten und die Backward Secrecy sichergestellt. Mithilfe des gehashten $\text{GKEK}_{\text{aktuell}}$ werden die neu erstellen Schlüssel GTEK_{neu} und GKEK_{neu} jeweils bitweise mit XOR überschlüsselt. Dem neuen Client versendet der GCKS das neue Schlüsselpaar (GTEK_{neu}, GKEK_{neu}) verschlüsselt mit dessen ausgehandeltem privaten KEK. Den restlichen Gruppenteilnehmern wird das neue Gruppenschlüsselpaar verschlüsselt mit dem bisherigen $\text{GKEK}_{\text{aktuell}}$ per Multicast zugesandt.

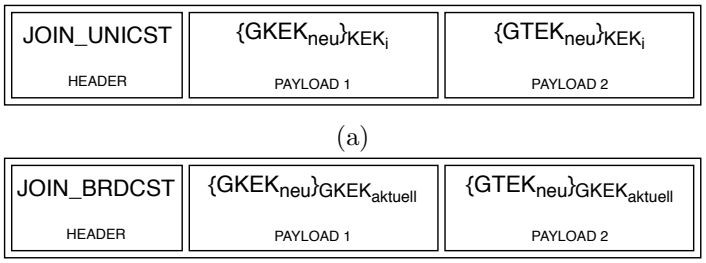

Abbildung 4.5: Die Datenpakete bei einem Eintritt für den neuen Gruppenteilnehmer (a) und für die restlichen Gruppenteilnehmer (b)

Bei einer Join Anfrage in CAKE sind aus Sicht des GCKS nur zwei Nachrichten zu verteilen: Eine Unicast an das neue Gruppenmitglied und eine Multicast an die bisherigen Mitglieder. Die Clients der spezifizierten Gruppe empfangen jeweils eine einzige Nachricht, worin die neuen Gruppenschlüssel enthalten sind. Der neue Teilnehmer entschlüsselt die empfangene Unicast Nachricht mit seinem privaten KEK und erhält die Gruppenschlüssel. Die restlichen Teilnehmer der Gruppe entschlüsseln das empfangene Paket mit dem bisherigen, in ihrem Besitz liegenden $\text{GKEK}_{\text{aktuell}}$.
Unauthorisierte Clients können weder die Unicast Nachricht noch die Multicast Nachricht entschlüsseln. Ihnen ist sowohl der persönliche KEK des neuen Teilnehmers als auch der $\text{GKEK}_{\text{aktuell}}$ nicht bekannt.

4.3.3 Austritt von Teilnehmern aus einer Gruppe - Leave

Ein Gruppenmitglied kann auf unterschiedliche Weise aus einer gesicherten Gruppenkommunikation aus dem CAKE System entfernt werden. Er kann eine Anfrage beim Gruppen-

Manager stellen, um eine Gruppe zu verlassen oder er wird vom CAKE GCKS selbst entfernt. Nachdem feststeht, dass ein Teilnehmer aus einer Gruppe ausgeschlossen wird, kann das bestehende Gruppenschlüsselpaar ($GTEK_{aktuell}$, $GKEK_{aktuell}$) nicht mehr verwendet werden, da der austretende Teilnehmer weiterhin im Besitz dessen ist. Damit also die Backward Secrecy gewährleistet und die bisherige mit dem $GTEK_{aktuell}$ übertragene Kommunikation geheimgehalten wird, muss das Gruppenschlüsselpaar aus $GTEK_{aktuell}$ und $GKEK_{aktuell}$ erneuert werden. Der Algorithmus des Leave Requests beruht wie bei der initialen Erzeugung einer Gruppe auf den CRT.
Im Gegensatz des initalen CRT gehen aber bei der Leave Operation nicht die Schlüssel jedes Teilnehmers in die Berechnung des CRT ein, sondern nur eine Teilmenge. Somit vermindern sich die Berechnungskosten für den GCKS des CAKE Systems, die in MANETs so gering wie möglich gehalten werden sollten.
Der GCKS verwaltet eine ternäre Baumstruktur, anhand dessen ein verkleinerter CRT berechnet werden kann. Für den Key Server verminden sich daduch Berechnungskosten. Für jede gesicherte Gruppenkommunikation im CAKE System speichert der GCKS eine unabhängige Baumstruktur. Sind die Gruppenteilnehmer über die entsprechenden Baumknoten informiert und haben sie Kenntnis darüber, mit welchem Schlüsselpaar (KEK, Primzahl m) das Secure Lock aufgelöst werden kann, wird nur eine einzige Multicast Nachricht seitens des GCKS benötigt, um die neuen Gruppenschlüssel chiffriert an die verbleibenden Gruppenteilnehmer zukommen zu lassen.

Im Folgenden wird der Schlüsselbaum beschrieben. Die Baumstruktur errinnert an den Algorithmus des LKH Verfahrens (siehe Abschnitt 2.4.1), bei dem ebenfalls ein Schlüsselbaum verwaltet wird.

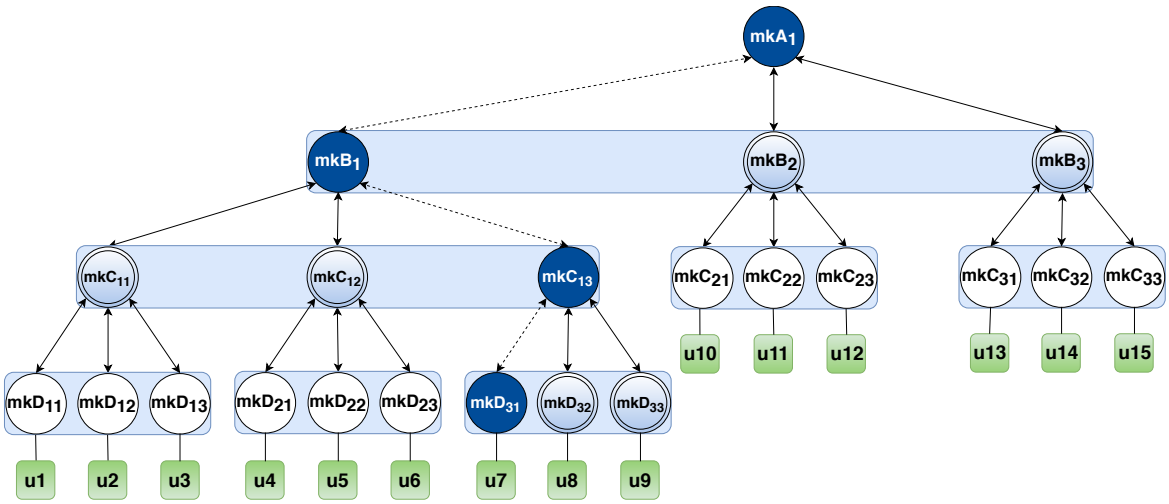

Abbildung 4.6: Der Schlüsselbaum im CAKE System - angepasst aus [12]

Jeder Knoten beinhaltet eine Primzahl und einen KEK, die beide in den CRT einfließen. Zusätzlich ist der Baum in Ebenen eingeteilt. Daher definiert die Bezeichnung *mkX* eine Primzahl m, einen KEK k und die Ebene X für jeden Knoten in der Baumstruktur. Informationen über Gruppenmitglieder werden nur als Blattknoten eingefügt. Jeder Blattknoten repäsentiert dementsprechend einen Gruppenteilnehmer (u1 bis u15).

Beim Austritt eines Teilnehmers u7 wird sein Pfad bis zur Wurzel markiert (in Abbildung 4.6 sind die entsprechenden Knoten dunkel markiert). Die markierten Knoten sind kompromittiert und werden nicht in die folgende CRT Berechnung miteinbezogen. Stattdessen gelangen pro Pfad die im Baum am weitesten oben und nicht markierten Knoten mkB_2, mkB_3, mkC_{11}, mkC_{12}, mkD_{32} und mkD_{33} in die Berechnungen des CRT. Diese befinden sich in jeder Ebene - außer in der Wurzelebene - neben den komorimittierten Knoten. In der Grafik sind diese jeweils doppelt eingerahmt. Dadurch verringert sich in diesem Fall der Rechenaufwand des CRT von 14 Knoten auf sechs.
Der GKEK fließt sechs mal in den CRT ein. Jedes Mal wird er dabei mit einem der sechs KEKs aus den doppelt eingerahmten Knoten verschlüsselt. Bei der Berechnung des CRT wird analog wie in Abbildung 4.3 veranschaulicht vorgegangen. Neben der Liste des verschlüsselten GKEK gehen zusätzlich die Primzahlen aus denselben sechs Knoten in den CRT ein. Das Ergebnis des CRT - das Secure Lock - stellt ein Teil des zu versendenden Datagramms dar. Außerdem wird der GTEK bitweise XOR mit dem GKEK überschlüsselt und vervollständigt den zweiten Teil des Pakets. Die Informationen über das neue Gruppenschlüsselpaar aus GTEK_{neu} und GKEK_{neu} können demnach mit folgendem Datagramm veranschaulicht werden.

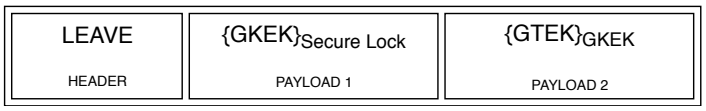

Abbildung 4.7: Datagramm bei der Austrittsoperation

Da die CRT Berechnung aus dem Schlüsselbaum heraus zeitintensiv und teuer ist, baut der GCKS immer nur dann die Baumstruktur auf, wenn keine anderen Berechnungen erforderlich sind. Allein der GCKS hat Kenntnis über die gesamte Baumstruktur und verwaltet sie.

Die aufgrund von Teilnehmeraustritten entstehenden freien Stellen im Schlüsselbaum können mit neuen Teilnehmern, die der Gruppe beitreten, aufgefüllt werden.

Bei Bedarf hat der GCKS die Möglichkeit, einen entarteten Baum neu aufzubauen und zu balancieren. Dies kann dann geschehen, wenn bereits mehrere Teilnehmer aus der gesicherten Gruppenkommunikation ausgetreten sind bzw. vom GCKS entfernt wurden und der GCKS keine anderen Aufgaben bewerkstelligen muss. Die Baumhöhe kann sich dabei reduzieren. Allerdings sollte dies nur dann geschehen, wenn der GCKS keine anderen, höher priorisierten Aufgaben bewältigen muss wie beispielsweise einen neuen Teilnehmer in eine Gruppe aufzunehmen.

4.3.4 Mehrfach Operationen

In eine Gruppe kann nicht nur ein einzelner Teilnehmer hinzugefügt werden, sondern mehrere Teilnehmer können gleichzeitig aufgenommen werden. Ebenso können mehrere Teilnehmer zugleich eine gesicherte Gruppenkommunikation verlassen bzw. entfernt werden. In den folgenden zwei Abschnitten werden diese beiden Aspekte beschrieben.

Mehrfach-Eintritt / Verschmelzung

Um eine Truppe zu verstärken, kann der Gruppenmanager anordnen, dass mehrere weitere Soldaten zu einer bestehenden Gruppe hinzugefügt werden. Dabei geht die Authorisierte Instanz (AI) in CAKE äquivalent wie beim Einzeleintritt vor [12]. Jedem neuen Mitglied

wird das neue Gruppenschlüsselpaar aus GTEK_{neu} und GKEK_{neu} verschlüsselt mit seinem persönlichen Schlüssel KEK_i zugesandt. Die restlichen Teilnehmer erhalten das Datenpaket via Multicast. Die darin enthaltenen neuen Gruppenschlüssel sind mit dem bisherigen $\text{GKEK}_{\text{aktuell}}$ chiffriert. Auf diese Weise werden aus Sicht des GCKS eine Multicast und x Unicast Nachrichten benötigt, wobei x die Anzahl der neu hinzugekommenen Teilnehmer bezeichnet.

Alternativ lassen sich die Schlüsselpaare aus Primzahl m_i und geheimen Schlüssel KEK_i aller neuen Teilnehmer über ein CRT zusammenfassen. Der GKEK_{neu} wird dabei jeweils mit den privaten KEKs der neuen Teilnehmer verschlüsselt und gemeinsam mit deren Primzahlen entsteht der Secure Lock. Das Datagramm kann anschließend mit dem mit GKEK_{neu} überschlüsseltem GTEK_{neu} addiert und per Multicast zugesandt werden. Für die restlichen, bisherigen Teilnehmer ändert sich der Erhalt des neuen Gruppenschlüssels nicht. Sie empfangen die neuen Schlüsselinformationen wie in der ersten Möglichkeit, bei der das neue Schlüsselpaar mit dem $\text{GKEK}_{\text{aktuell}}$ verschlüsselt wird.

Bei der Verschmelzung zweier oder mehrerer bestehender Gruppen wird ein neues Gruppenschlüsselpaar (GTEK_{neu}, GKEK_{neu}) erstellt. Die neuen Schlüssel werden mithilfe des bestehenden $\text{GKEK}_{\text{aktuell}}$ der zu verschmelzenden Gruppen jeweils verschlüsselt und an jede Gruppe verteilt. So kann jede Gruppe das neue Gruppenschlüsselpaar mit ihrem aktuellen GKEK entschlüsseln. Es werden entsprechend der Anzahl der zu verschmelzenden Gruppen Nachrichten benötigt. Für drei zu verschmelzenden Gruppen werden drei Multicast Nachrichten benötigt. Backward Secrecy ist dabei sichergestellt, da die Teilnehmer nicht die Möglichkeit haben, an den $\text{GKEK}_{\text{aktuell}}$ der anderen Gruppen heranzukommen.

Mehrfach-Austritt / Aufteilung

Vom Gruppenmanager kann ebenso angeordnet werden, mehrere Soldaten zugleich aus einer Gruppe zu entnehmen, um ihnen jeweils separate Aufgaben zuzuweisen. Dabei unterscheidet sich der Algorithmus, den der GCKS verrichten muss, kaum vom Einzelaustritt. Hierzu werden in der ternären Baumstruktur entsprechend mehrere Pfade markiert. Der restliche Vorgang geschieht wie beim Einzelaustritt in 4.3.3.

Wird eine Gruppe in zwei oder mehreren Gruppen aufgeteilt, wird für jede neu entstehende Gruppe jeweils eine Multicast benötigt. Darin enthalten ist jeweils die Lösung des CRT Systems, der Secure Lock. Die Teilnehmer der jeweiligen Gruppen sind nicht in der Lage, die Kommunikation der anderen Gruppen zu lesen.

4.3.5 Re-Key einer Gruppe

Für das Re-Keying des GTEK einer Gruppe existieren zwei Möglichkeiten. Für die erste Variante generiert der GCKS ein neues Gruppenschlüsselpaar aus GTEK_{neu} und GKEK_{neu}) welche jeweils mit dem $\text{GKEK}_{\text{aktuell}}$ verschlüsselt wird. Das somit entstandene Datagramm (Abbildung 4.8a) wird im Anschluss per Multicast an alle Gruppenteilnehmer verschickt.

Durch das Festlegen eines klar definierten Zeitpunktes können die Gruppenmitglieder synchron auf den neuen GTEK_{neu} umstellen.

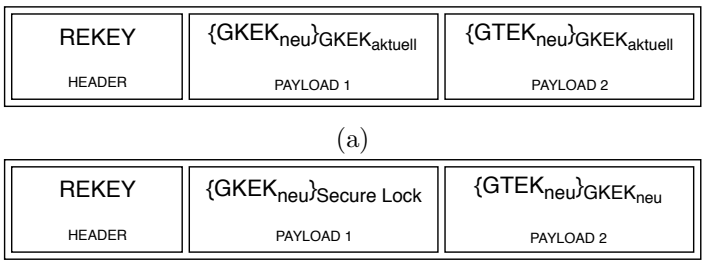

Abbildung 4.8: (a) Die effizientere und (b) die rechenintensive Variante beim Rekey im Vergleich

Die zweite Option (Abbildung 4.8b) sieht die Nutzung der beiden Geheimnisse Primzahl m_i und KEK_i der Teilnehmer bzw. des CRT Systems vor. Dabei verfährt der GCKS entsprechend des Konzeptes der initialen Erzeugung einer Gruppe bzw. des ersten GTEK.

In der Implementierung dieser Arbeit wird die erste Variante übernommen, da dies einen reduzierten Rechenaufwand für den GCKS darstellt. Im Zusammenhang mit dem gestellten Szenario muss die Rechenpower effizient genutzt und die Gruppenschlüssel möglichst rasch versandt werden.

Unabhängig, welche Option gewählt wird, muss zwingend ein neues Gruppenschlüsselpaar aus GTEK_{neu} und GKEK_{neu} generiert werden. Für die erste Variante (Abbildung 4.8a) müssen diese jeweils mit dem $\text{GKEK}_{\text{aktuell}}$ bitweise mittels XOR verschlüsselt werden. Bei der Wahl der zweiten Möglichkeit müssen jedoch im Kontext des CRT mehrere Multiplikationen und Modulo Berechnungen durchgeführt werden. Zusätzlich ist eine XOR Operation notwendig, um den GTEK_{neu} abzusichern und zum Datagramm hinzuzufügen.

Empfängt ein Client das Datagramm wie in Abbildung 4.8a veranschaulicht, so erhält er mit zwei XOR Operationen die neuen Gruppenschlüssel. Empfängt er Datagramm 4.8b, entschlüsselt er das Secure Lock, indem er darauf die Modulo Funktion mit seinem persönlichen KEK anwendet. So erhält er den mit seinem KEK verschlüsselten GKEK_{neu}. Mithilfe des entschlüsselten GKEK kann die XOR Funktion auf den zweiten Teil des Datagramms angewendet und der GTEK_{neu} errechnet werden. Auf Seiten des Clients besteht genauso wie beim GCKS ein erhöhter Rechenaufwand, wenn die zweite Option gewäht wird.

Für den GCKS ist bei beiden Varianten der Nachrichtenaufwand der Gleiche. Es ist jeweils eine Multicast Nachricht notwendig. Aus Sicht der Gruppenmitglieder stellt die Wahl der Datagramme in der Anzahl der empfangenen Nachrichten ebenfalls keinen Unterschied. Jeder Gruppenteilnehmer empfängt ein Rekey Paket. Der Rechenaufwand jedoch ist, wie beschrieben, für beide Parteien vorteilhafter, wenn die erste Variante gewählt wird.

5 Implementierung

——— Der folgende Text ist ein direktes Zitat von [7]. ———

ür die Implementierung des Protokolls wurde das in Kapitel 2.2 beschriebene Betriebssystem RIOT gewählt. Als Programmiersprache dient C. Anders als vom Design vorgesehen, werden bei der Implementierung die kryptographischen Algorithmen nicht von Client und GCKS ausgehandelt, sondern im Vorfeld definiert. Dies hat den Hintergrund, dass die Berechnung und Verteilung der Gruppenschlüssel anhand des *Chinese Remainder Theorems* Hauptmerkmal dieser Arbeit ist.

5.1 Datenstrukturen

Für die Funktionsweise des Protokolls ist die Definition einiger Datenstrukturen notwendig, welche beispielsweise das Abbilden von Clients und Gruppen erlauben.

5.1.1 Nachrichten

Die vom GNRC Network Stack verwendeten, und im *pktbuf* zwischengespeicherten Nachrichten bestehen aus einer verketteten Liste. Die Elemente dieser Liste bestehen aus Paket „Stückchen“ (*packet snips*) und werden durch das von RIOT definierte Struct *gnrc_pktsnip_t* abgebildet. Jedes dieser Structs stellt entweder einen Header oder eine Nutzlast dar. Abbildung 5.5 stellt diesen Aufbau am Beispiel eines UDP Pakets dar.

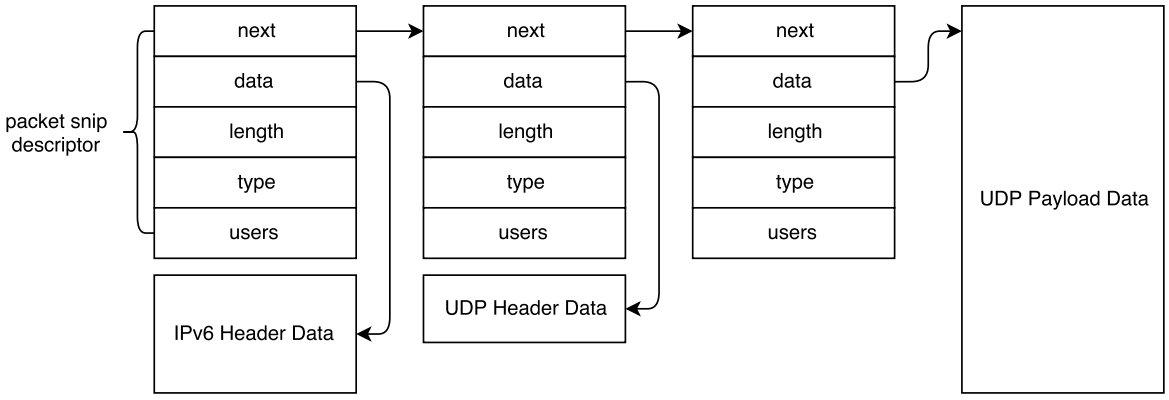

Abbildung 5.1: Aufbau eines GNRC Pakets

Das Hinzufügen eines *gnrc_pktsnip_t* in den *pktbuf* erfolgt über den Aufruf der Funktion *gnrc_pktbuf_add()*. Diese erwartet das nächste Element des Pakets, einen Pointer auf die Daten, die Länge der Daten und den Protokolltyp als Parameter und gibt einen Pointer auf

die Speicheradresse im *pktbuf* zurück. Hierbei wird das erste ausreichend große, freie Segment im Puffer ausgewählt, wodurch durchaus eine Fragmentierung des *pktbuf* entstehen kann. Die empfangenen Pakete bestehen aus einer verketteten Liste mit nur einem Element, welches die gesamten Daten beinhaltet. Durch den Aufruf Funktion *gnrc_pktbuf_mark()* kann dieses Element geteilt werden, wobei die ersten n Bytes des Pakets als nächstes Element an das ursprüngliche Element angehängt wird. Dadurch kann das Paket für die weitere Verarbeitung in die einzelnen Header und Payloads aufgetrennt werden. Wurde eine Nachricht bearbeitet oder wird sie nicht länger benötigt, kann der im *pktbuf* belegte Speicher mithilfe der Funktion *gnrc_pktbuf_release()* wieder freigegeben werden.

——— Ende des Zitats. ———

5.1.2 Ternäre Baumstruktur

Die Verschlüsselung und Verteilung des neuen Gruppenschlüssels bei der Austrittoperation erfordert eine CRT Berechnung, in der im Gegensatz zum initialen CRT bei der Gruppenerzeugung nicht das geheime Schlüsselpaar *(m_i, KEK_i)* jedes Teilnehmers miteinfließen, sondern nur eine Teilmenge und zusätzlich entsprechende Knoten aus der Baumstruktur, die im Kapitel 4.3.3 beschrieben wurde.

Das Softwaredesign der Baumstruktur ist entscheidend für den Verwaltungsaufwand und für den damit verbundenen unterschiedlichen Nachrichteninhalt innerhalb des Protokolls. In diesem Kapitel werden zwei Paradigmen vorgestellt, wie neue Gruppenteilnehmer im Schlüsselbaum eingefügt werden können. In der Implementierung dieser Arbeit wurde zunächst mit dem ersten Paradigma angefangen und Zweiteres wurde schließlich ausgewählt. Wie in Kapitel 4.3.3 erläutert, werden Gruppenteilnehmer - unabhängig von der Einfügestrategie - als Blattknoten repräsentiert und nicht als Zwischenknoten.

In der ersten Idee werden neue Knoten wie beim B-Baum in die Tiefe eingefügt. Dabei verändert sich der Wurzelknoten nicht, vielmehr wächst der Baum von der Wurzel aus nach unten in die Tiefe, wie in Abbildung 5.2 dargestellt.

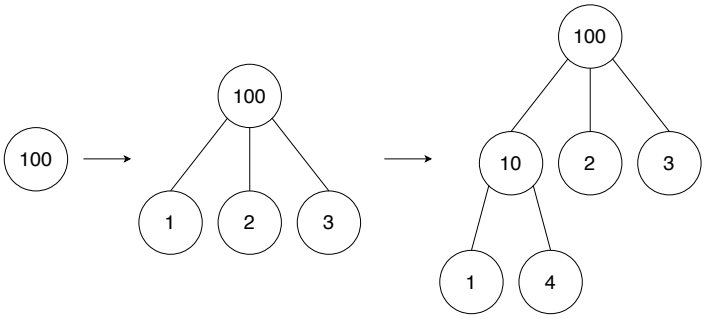

Abbildung 5.2: Neue Elternknoten für Clientknoten durch das Paradigma des B-Baum und der damit verbundene erhöhte Verwaltungsaufwand

Für jedes Gruppenmitglied wird ein neuer Blattknoten eingefügt, in dem die Primzahl und der persönliche KEK des Teilnehmers beinhaltet sind. Nach der Generierung der Wurzel können die ersten drei Blattknoten daran angehängt werden (veranschaulicht in Abbildung 5.2). Kommt ein vierter Client hinzu, müsste einer der drei Clientknoten zu einem Zwischenknoten (in der Grafik die Nummer 10) umgewandelt werden, dem eine neue Primzahl und ein neuer KEK zugewiesen wird.

An dem neuen Zwischenknoten 10 kann dann der Clientknoten 1, dessen Position der neue Zwischenknoten 10 eingenommen hat, als dessen Kindknoten angehängt werden. Auf diese Weise ändert sich der Vaterknoten des Clients mit der Nummer 1 (siehe Abbildung 5.2). Nachdem aufgrund weiterer Einfügungen die Clients 2 und 3 ebenfalls zu Zwischenknoten umgewandelt wurden und jeweils drei Clients an allen drei Zwischenknoten eingefügt wurden, müsste beim nächsten Einfügen wieder einer der Clients aus der untersten Ebene zu einem Zwischenknoten umgewandelt und das gleiche Prozedere wie beschrieben durchgegangen werden. Das bedeutet, bei jeder vollständig aufgefüllten Ebene, in der sich nur Clientknoten befinden, muss beim nächsten Einfügen wieder ein Clientknoten umgewandelt werden.

Der Nachteil einer solchen Einfügemethode ist, dass bei jedem Einfügen ein bereits sich im Baum befindenden Client zu einem Zwischenknoten umgewandelt werden muss. Dieser muss zusätzlich per Unicast über seinen Pfad zur Wurzel informiert werden. Bei bereits fortgeschrittenen Baumstrukturen mit mehreren Ebenen und mit dementsprechend vielen Clients besteht ein hoher Verwaltungsaufwand und das Einfügen wird komplex.

Eine geeignetere Baumstruktur ist das Paradigma des B^+-Baumes, das teilweise in die Baumstruktur dieser Arbeit eingegliedert wurde. Beim einem Overflow wird nicht wie beim anfangs beschriebenen Paradigma ein Kindknoten umgewandelt und in die Tiefe eingesetzt, sondern eine neue Wurzel (in Abbildung 5.3 mit der Nummer 100) erstellt und daran verwiesen.

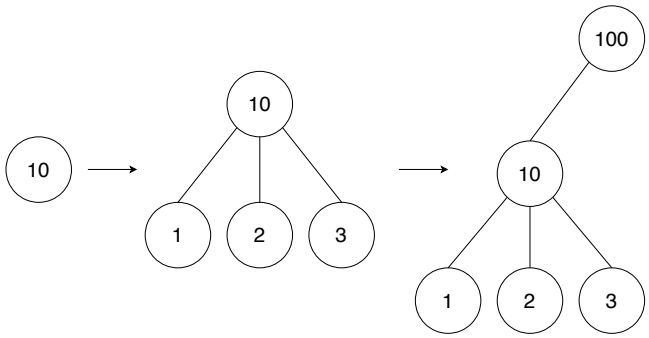

Abbildung 5.3: Beim Paradigma des B^+-Baum werden direkte Elternknoten der Blätter beibehalten.

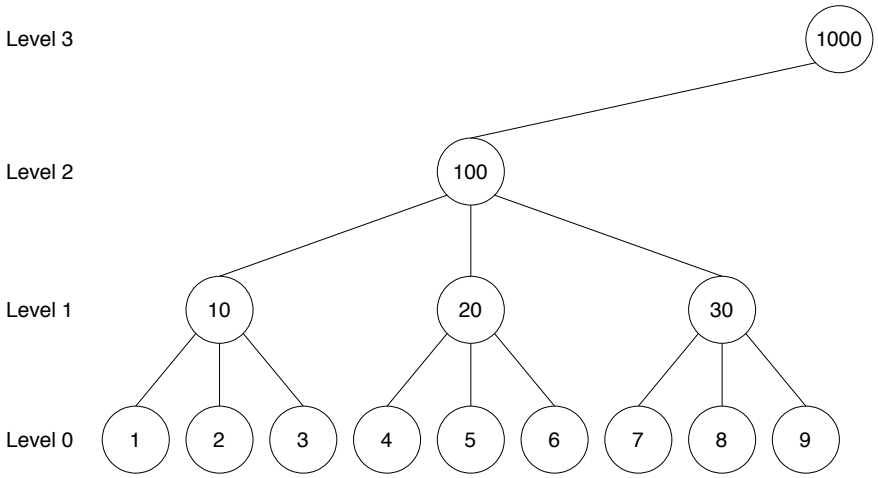

Abbildung 5.4: Die Baumstruktur, nachdem neun Clients eingefügt wurden.

Sobald eine neue Wurzel erstellt wurde, sollten aufgrund der gleichen Problematik wie oben beschrieben, die zwei restlichen freien Plätze der Wurzel nicht mit Clientknoten besetzt werden. Stattdessen werden von der Wurzel aus so lange neue Zwischenknoten angelegt, bis Zwischenknoten mit Level 1 erstellt wurden. An diesen können anschließend Kindknoten eingefügt werden.

Dadurch, dass bei einem Overflow einer Ebene eine neue Wurzel (in Abbildung 5.4 der Knoten mit Nummer 1000) angelegt wird, ist es möglich, allen Teilnehmern des in der Abbildung links dargestellten Teilbaumes über eine Multicast über die neue Wurzel zu informieren.

Der *ungünstigste Fall*, jeden Teilnehmer einzeln per Unicast anzusprechen, tritt beim ersten Austritt auf. Bei der ersten Leave-Operation einer Gruppe haben die Gruppenteilnehmer nämlich keinerlei Kenntnis über ihre Zwischenknoten und ihren Pfad bis zur Wurzel.

In der Implementierung dieser Arbeit wird bei der Austritt Operation der Fokus auf den *ungünstigsten Fall* gelegt. Den Gruppenteilnehmern wird jeweils per Unicast mitgeteilt, mit welchem Knoten sie jeweils das CRT auflösen und dadurch den GKEK erhalten können.

5.2 Konfiguration

Die Quelldatei kann vor dem Kompilieren und Ausführen über eine Konfigurationsdatei angepasst und dabei verschiedene Parameter verändert werden. So kann zum einen das aus einer Zeichenkette bestehende PSK in eine beliebig andere verändert werden. Zum anderen lassen sich Parameter wie die maximale Anzahl der Clients, die eine Gruppe aufnehmen kann, einstellen. Die Anzahl der Gruppen selbst lässt sich ebenfalls regulieren.
Die Portnummer kann über die Variable `CAKE_PORT` gesetzt werden. Wie bereits in Kapteil 4 beschrieben, wird standardmäßig die Portnummer 849 verwendet.

Außerdem können die Längen veschiedener Schlüssel eingestellt werden.

- Die Bytelänge der im CAKE System notwendigen Schlüssel wie GTEK, GKEK und KEK können mit `DEF_KEY_LENGTH` gesetzt werden.
- Die Länge der für jeden Client zugewiesenen und für jeden Baumknoten erstellte Primzahl kann mit `PRIME_LENGTH` verändert werden. Hierbei sollte beachtet werden, dass die Primzahl stets größer sein sollte als die in den CRT gelangte KEK Variable, damit die Berechnungen des CRT (siehe Kapitel 5.5) funktionieren. Standardmäßig hat `PRIME_LENGTH` den Wert `DEF_KEY_LENGTH + 1`.

——— Der folgende Text ist ein direktes Zitat von [7]. ———

Nachrichten werden über den Aufruf der *cake_receive()* Funktion empfangen. Da diese Funktion das Programm solange blockiert, bis eine Nachricht empfangen wurde, muss diese in einen zweiten Thread ausgelagert werden, um weiterhin eine Kommunikation zwischen Client und GCKS zu ermöglichen.

```
1 static msg_t _rcv_msg_queue[MAIN_QUEUE_SIZE];
2 char rcv_thread_stack[THREAD_STACKSIZE_MAIN];
3
4 void *rcv_thread(void *arg)
```

```
{
    (void) arg;

    msg_init_queue(_rcv_msg_queue, MAIN_QUEUE_SIZE);

    gnrc_netreg_entry_t receive_netreg = GNRC_NETREG_ENTRY_INIT_PID(
        GNRC_NETREG_DEMUX_CTX_ALL, sched_active_pid);
    receive_netreg.demux_ctx = CAKE_PORT;
    gnrc_netreg_register(GNRC_NETTYPE_UDP, &receive_netreg);

    while (1) {
        gnrc_pktsnip_t *msg = cake_receive();
        msg_handle(msg);
        gnrc_pktbuf_release(msg);
    }

    return NULL;
}

thread_create(rcv_thread_stack, sizeof(rcv_thread_stack),
              THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST,
              rcv_thread, NULL, "rcv_thread");
```

Code 5.1: Erstellung eines zweiten Threads

Um Nachrichten empfangen zu können, müssen die jeweiligen Threads zuerst einen entsprechenden Eintrag im Netzwerk Register (*netreg*) hinterlegen. Dazu wird die Thread ID sowie ein *demux*-Kontext benötigt. Über den *demux*-Kontext lassen sich Pakete des selben Typs weiter unterscheiden. Da das Protokoll auf UDP basiert, ist der *demux*-Kontext die Portnummer.

```
client_info_t clients[MAX_CLIENT_COUNT] = {{0}};
group_info_t groups[MAX_GROUP_COUNT] = {{0}};
```

Code 5.2: Initialisierung der Client- und Gruppen-Arrays

Der für die Client und Gruppen Strukturen benötigte Speicher wird in Abhängigkeit der Konstanten *MAX_CLIENT_COUNT* und *MAX_GROUP_COUNT* alloziiert.

——— Ende des Zitats. ———

5.3 Nachrichtenverarbeitung

——— Der folgende Text ist ein direktes Zitat von [7]. ———

m die in diesem Protokoll versendeten Nachrichten korrekt zuzuordnen und verarbeiten zu können, wird zunächst ein weiteres Struct definiert, durch das ein eigener Headertyp abgebildet wird. Das Feld *exchange_type* gibt hierbei an, um welchen Nachrichtentyp des Protokolls es sich handelt.

```c
typedef struct cake_hdr {
    uint8_t exchange_type;
    payload_type_t pl_next;
    network_uint16_t msg_id;
    network_uint16_t length;
} __attribute__ ((packed)) cake_hdr_t;
```

Code 5.3: CAKE Header Struct

Beim Empfangen einer Nachricht wird zunächst überprüft, welchen Typ diese Nachricht hat, damit entsprechend darauf reagiert werden kann. Dies erfolgt über die *msg_handle* Funktion. Ein Auschnitt der Funktion wird in Code 5.4 gezeigt.

```c
uint8_t msg_handle(gnrc_pktsnip_t *msg)
{
    size_t len = msg->size;
    cake_hdr_t *hdr = cake_hdr_handle(msg);

    if (hdr == NULL) {
        DEBUG("ERROR: Could not parse CAKE header");
        return EXIT_ERROR;
    }

    if (byteorder_ntohs(hdr->length) != len) {
        DEBUG("ERROR: Invalid packet length");
        return EXIT_ERROR;
    }

    if (hdr->exchange_type == CAKE_INIT && client.state == STATE_IDLE) {
        if (!handle_init_response(msg)) {
            return EXIT_ERROR;
        }
    }
    else if (hdr->exchange_type == CAKE_AUTH && client.state == STATE_INIT) {
        if (!handle_auth_response(msg)) {
            return EXIT_ERROR;
        }
    }
    else if (hdr->exchange_type == CAKE_GROUP_JOIN &&
             client.state == STATE_AUTH) {
        if (!handle_join_group_response(msg)) {
            return EXIT_ERROR;
        }
    }

    // [...]

    else {
        DEBUG("ERROR: Unknown Exchange Type - aborting!");
        return EXIT_ERROR;
    }

    return EXIT_SUCCESS;
}
```

Code 5.4: *msg_handle* Funktion des Clients

Wurde ein gültiger Nachrichten Typ ermittelt, ruft der Client die entsprechende *handle*-Funktion auf. Der GCKS verhält sich beim Empfangen einer Nachricht analog dazu.

An das *packet snip*, das den CAKE Header enthält, werden nacheinander die einzelnen Payloads angehangen. Aufgrund der Tatsache, dass die Nachrichten als ein einziges Datagramm empfangen werden, bedarf es eine Lösung, wie man die Daten wieder auftrennen kann. Jeder Payload wird daher mit einem eigenen Header versehen, der die Länge der Daten, sowie den Typ des nächsten Payloads enthält.

```
typedef struct payload_hdr {
    payload_type_t pl_next;
    network_uint16_t length;
} __attribute__ ((packed)) payload_hdr_t;
```

Code 5.5: Payload Header Struct

5.4 Initialisierung und Authentifizierung

Der Client wird mittels der *init_client_info()* initialisiert. Hierbei wird unter die IPv6 Adresse sowie eine eindeutige, auf der Seriennummer der CPU basierende, ID ausgelesen. Zusätzlich dazu werden die für den Diffie-Hellman-Schlüsselaustausch benötigten Werte generiert.

```
client_info_t client_info;

uint8_t init_client_info(void)
{
    client_info.state = STATE_IDLE;
    client_info.id = get_luid();
    client_info.addr = get_ipv6_from_ifs();
    client_info.dh_ctxt = init_dh_ctxt();
}
```

Code 5.6: Initialisierung des Clients

```
dh_context_t *init_dh_ctxt(void)
{
    dh_ctxt.curve = uECC_secp256r1();
    dh_ctxt.public_size = uECC_curve_public_key_size(dh_ctxt.curve);
    dh_ctxt.private_size = uECC_curve_private_key_size(dh_ctxt.curve);
    dh_ctxt.compressed_public_size = dh_ctxt.private_size + 1;

    if (!uECC_make_key(dh_ctxt.public, dh_ctxt.private, dh_ctxt.curve)) {
        return NULL;
    }

    uECC_compress(dh_ctxt.public, dh_ctxt.compressed_public, dh_ctxt.curve);
```

```
13
14      return &dh_ctxt;
15  }
```

Code 5.7: Generierung der Diffie-Hellman-Werte

Die Registrierung des Clients am GCKS erfolgt über den Aufruf der zwei Funktionen *cake_init()*, welche den Diffie-Hellman-Schlüsselaustausch implementiert, und *cake_auth()*, wobei sich Client und GCKS gegenseitig mittels des PSK authentifizieren.

5.5 Berechnung des CRT

Da bei der Berechnung des CRTs, in Abhängigkeit von der Schlüssellänge des GKEKs, der Länge der Primzahlen, sowie der Anzahl an Clients über die das CRT erstellt wird, mitunter sehr große Zahlen entstehen können, Standard C aber keinen Datentyp für Zahlen enthält, der mehr als 64 Bit darstellen kann, muss eine externe Bibliothek eingesetzt werden. Hierbei handelt es sich die GNU Multiple Precision Arithmetic Library (GMP), beziehungsweise eine Teilmenge dieser Bibliothek, die durch die Datei *mini-gmp.c* bereitgestellt wird. GMP wird in Kapitel 5.8 tiefergehend behandelt.

Wie vom Protokoll festgelegt, wird zunächst der GKEK der Gruppe mit dem KEK eines jeden Clients verschlüsselt. Der verschlüsselte GKEK und die Primzahl des Clients werden im Anschluss in einen GMP Integer umgewandelt.

```
1  mpz_t primes[n];
2  mpz_t encr_gkeks[n];
3
4  uint8_t encr_gkek[DEF_KEY_LENGTH] = {0};
5
6  for (int i = 0; i < n; i++) {
7      // Initialisierung der mpz_t Variablen
8      mpz_init(primes[i]);
9      mpz_init(encr_gkeks[i]);
10
11     // XOR-Verschlüsselung des GKEK mit dem KEK eines Clients
12     xor_encr(gkek, DEF_KEY_LENGTH, encr_gkek, clients[i]->keymat.key);
13
14     // Einlesen und Umwandeln des verschlüsselten GKEKs in einen GMP Integer
15     mpz_import(encr_gkeks[i], DEF_KEY_LENGTH, 1, 1, 0, 0, encr_gkek);
16
17     // Einlesen und Umwandeln der Primzahl des Clients in einen GMP Integer
18     mpz_import(primes[i], PRIME_LENGTH, 1, 1, 0, 0, clients[i]->prime);
19 }
```

Code 5.8: Umwandlung der GKEKs und Primzahlen

Das CRT kann nun mittels der Funktion *chinese_remainder()* berechnet werden. Als Parameter werden ihr der Pointer des Speicherbereichs übergeben, in das das Ergebnis geschrieben werden soll, jeweils ein Pointer auf die Arrays der verschlüsselten GKEKs und Primzahlen, sowie die Anzahl der Elemente, über die das CRT berechnet werden soll.

```
void chinese_remainder(mpz_t *crt, mpz_t encr_gkeks[], mpz_t primes[], size_t
    n)
{
  mpz_t prod, quot, sum, tmp;

  // Initialisierung der mpz_t Variablen
  mpz_init_set_ui(prod, 1);
  mpz_init_set_ui(sum, 0);
  mpz_init(quot);
  mpz_init(tmp);

  // Berechnung des Produkts über alle Primzahlen
  mpz_mul_arr(prod, primes, n);

  for (size_t i = 0; i < n; i++) {
    mpz_cdiv_q(quot, prod, primes[i]);
    mpz_invert(tmp, quot, primes[i]);
    mpz_mul(tmp, tmp, encr_gkeks[i]);
    mpz_addmul(sum, tmp, quot);
  }

  mpz_mod(sum, sum, prod);

  mpz_set(*crt, sum);

  // Freigabe des belegten Speichers
  mpz_clear(prod);
  mpz_clear(quot);
  mpz_clear(sum);
  mpz_clear(tmp);
}
```

Code 5.9: *chinese_remainder()* Funktion

Die hierbei entstehende Zahl, das SL, wird im Anschluss als Zeichenkette abgelegt, damit es an die Clients verschickt werden kann. Hierzu wird zunächst über den Aufruf der Funktion *mpz_sizeinbase()* die Anzahl der Stellen des SL ermittelt und ein Byte-Array in der entsprechende Größe initialisiert. Hierbei muss beachtet werden, dass zu dem Rückgabewert der *mpz_sizeinbase()* Funktion der Wert 2 hinzu addiert werden muss, da die Funktion *mpz_get_str()* zwei Elemente des Arrays für ein potentielles Minuszeichen sowie den Nullterminator beansprucht (Code 5.10).

```
size_t crt_length = mpz_sizeinbase(crt, 10) + 2;

uint8_t crt_str[crt_length];
mpz_get_str((char *) crt_str, 10, crt);
```

Code 5.10: Umwandlung des CRTs in eine Zeichenkette

Diese wandeln die Zeichenkette wieder in einen GMP Integer um und führen die Umkehrfunktion aus. Das Ergebnis dieser Operation ist der mit dem privaten Schlüssel des Clients

verschlüsselte GKEK. Dieser wird vom Client entschlüsselt (Code 5.11, woraufhin er sich im Besitz des Gruppenschlüssels, dem GTEK, befindet.

```
mpz_t crt, prime, tmp_gkek;

mpz_init(crt);
mpz_init(prime);
mpz_init(tmp_gkek);

// Einlesen des erhaltenen Strings
mpz_set_str(crt, crt_str, 10);

// Einlesen und Umwandeln der Primzahl des Clients in einen GMP Integer
mpz_import(prime, PRIME_LENGTH, 1, 1, 0, 0, client_prime);

// Modulo Berechnung
mpz_mod(tmp_gkek, crt, prime);

// Exportieren als Byte-Array
mpz_export(gkek, NULL, 1, 1, 0, 0, tmp_gkek);

mpz_clear(tmp_gkek);
mpz_clear(prime);
mpz_clear(crt);

// Entschlüsselung des GKEKs mit dem KEK
xor_encr(gkek, DEF_KEY_LENGTH, gkek, client_kek);
```

Code 5.11: Umwandlung und Lösen des CRTs, Entschlüsselung des GKEK

——— Ende des Zitats. ———

5.6 Gruppenoperationen

——— Der folgende Text ist ein direktes Zitat von [7]. ———

eben der Mlglichkeit eine neue Gruppe anzufordern, unterstützt die Implementierung einige der in Kapitel 4 beschriebenen Gruppenoperationen. Diese sind der Einzeleintritt, der Einzelaustritt, sowie das Re-Keying. Hierzu schickt der Client dem GCKS einen Payload mit der der entsprechenden Gruppen ID, auf die die Operation angewandt werden soll. Durch die Gruppen ID kann der GCKS die Operation eindeutig einer Gruppe zuordnen kann. Die Art der auszuführenden Operation wird im CAKE Header angegeben.

Erstellen einer Gruppe

Über die Funktion *cake_create_group()* kann ein Client den GCKS dazu auffordern eine neue Gruppe zu erstellen. In der Funktion *new_group()* kontrolliert der GCKS zunächst ob sich in dem Gruppen-Array noch ein freies Element befindet. Ist dies der Fall, erstellt er einen zufälligen GKEK und GTEK, generiert eine ID für die Gruppe und überprüft diese auf Einzigartigkeit.

```
group_info_t *new_group(group_info_t *groups)
{
    group_info_t *group = find_free_group(groups);

    if (group == NULL) {
        return NULL;
    }

    generate_keypair(group->keys.gtek, group->keys.gkek);

    uint8_t gid;
    do {
        cake_random(&gid, sizeof(gid));
    } while (!gid_is_unique(gid, groups));

    group->id = gid;

    return group;
}
```

Code 5.12: Erstellen einer neuen Gruppe mittels *new_group()*

Im weiteren Verlauf des Programms wird dem Client diese ID in einem entsprechendem Payload zurückgesendet. Der Client trägt die ID seinerseits in seinem Gruppen-Array ein.

Einzeleintritt

Für den Gruppenbeitritt steht dem Client die Funktion *cake_join_group()* zur Verfügung. Als Parameter wird die ID der Gruppe, der beigetreten werden soll, übergeben. Nach einem erfolgreichen Beitritt wartet der Client darauf, dass der GCKS ihm das Schlüsselmaterial zusendet.

Der GCKS überprüft zunächst ob eine Gruppe mit entsprechender ID existiert und fügt den Client hinzu, falls dies der Fall ist. Der weitere Programmablauf hängt in erster Linie davon ab, ob die Gruppenschlüssel bereits erstmalig verteilt wurden.

Ist dies nicht der Fall, muss der GCKS überprüfen ob bereits ausreichende Teilnehmer vorhanden sind. Über die Konfigurationsdatei kann festelegt werden, ab dem wievielten Client das CRT für die erste Schlüsselverteilung berechnet wird. Wurde diese Anzahl durch den Eintritt des Clients nicht erreicht, wartet der GCKS auf weitere Teilnehmer. Wurde die Mindestanzahl an Gruppenmitgliedern hingegen erreicht, berechnet der GCKS das CRT nach dem in Kapitel 5.5 beschriebenen Verfahren und verteilt es zusammen mit dem durch den GKEK verschlüsselten GTEK.

Wurde hingegen bereits Schlüsselmaterial verteilt, wird kein CRT berechnet, sondern ein neues, zufälliges Schlüsselpaar generiert. Die bereits vorhandenen Teilnehmer der Gruppe erhalten das neue Schlüsselpaar mittels einer Broadcast Nachricht, die die mit dem bereits bekannten GKEK verschlüsselten neuen Schlüssel enthält als Payload enthält. Der beigetretene Client erhält das neue Schlüsselpaar vie Unicast. Dieses wird zuvor mit dem seinem privaten Schlüssel (KEK) verschlüsselt.

```
uint8_t gkek_old[DEF_KEY_LENGTH];
memcpy(gkek_old, group->keys.gkek, DEF_KEY_LENGTH);

generate_keypair(group->keys.gtek, group->keys.gkek);

uint8_t encr_keys[2 * DEF_KEY_LENGTH];
xor_encr(group->keys.gtek, 2 * DEF_KEY_LENGTH, encr_keys, gkek_old);

send_keypair_multicast(group, encr_keys);
send_keypair_to_client(client->addr, &group->keys);
```

Code 5.13: Schlüsselverteilung

——— Ende des Zitats. ———

Einzelaustritt

Ein Client kann mittels der Funktion *cake_leave_group()* beim GCKS anfragen, eine Gruppe zu verlassen. Wie beim Einzeleintritt wird auch hierbei die ID der Gruppe, aus der der Teilnehmer austreten möchte, versendet. Der GCKS überprüft, ob es sich beim anfragenden Client um einen bereits im System registrierten und in der Gruppe sich befindenden Client handelt. Durchläuft dies positiv, sendet der GCKS dem anfragenden Client durch eine Unicast Nachricht die Bestätigung, dass dieser aus der Gruppe entfernt wird.

```
send_leave_response(requester, group_id, CLIENT_LEFT_GROUP);

// Backward Secrecy: New Group Keys
generate_keypair(group->keys.gtek, group->keys.gkek);

build_tree(group);
mark_client_path_to_root(requester_node, group);
collect_crt_inputs_from_tree(mpz_inputs, group->tree.root);

chinese_remainder(&secure_lock, mpz_keks, mpz_primes, crt_index);

for (size_t i = 0; i < MAX_CLIENT_COUNT; i++) {

    if (group->clients[i] != NULL && group->clients[i] != requester_node) {
        client_info_t *current_client = group->clients[i];
        node_t *client_node = find_client_node(group, current_client);
        // find relevant node for each client to decrypt the Secure Lock
        node_t *node = find_highest_possible_node(client_node);
        // send the node which the Client needs to encrypt the new Group Keys
        send_leave_unicast(group, crt_str, node, current_client, crt_length);
    }
}
```

Code 5.14: Algorithmus im Rahmen eines Austritts

Anschließend wird aufgrund der Backward Secrecy ein neues, zufälliges Gruppenschlüsselpaar ($GKEK_{neu}$, $GTEK_{neu}$) generiert. Der Pfad des austretenden Clients wird in der Baumstruktur markiert. Dann werden die für den CRT relevanten Knoten aus dem Baum aufgesammelt, anhand deren KEKs und Primzahlen der Secure Lock berechnet wird. Die restlichen

Clients aus der Gruppe werden jeweils per Unicast über den Secure Lock sowie den entsprechenden Knoten, mit dem sie den SL auflösen können, informiert. Innerhalb der Funktion *send_leave_unicast()* wird somit das Datenpaket des mit dem $GKEK_{neu}$ überschlüsselten $GTEK_{neu}$ gebildet. Daran wird der Secure Lock und der entsprechende Knoten, mithilfe deren der jeweilige Client den $GKEK_{neu}$ erhält, hinzugefügt. Außerdem wird die Gruppen ID angehängt. Auf diese Weise kommen die entsprechenden Clients der spezifizierten Gruppe in Besitz des neuen Gruppenschlüssels.
Die Informationen des anfragenden Clients werden anschließend aus der Gruppe gelöscht.

Re-Key

——— Der folgende Text ist ein direktes Zitat von [7]. ———

a nur der Group Manager dazu berechtigt ist, den Re-Key Prozess manuell einzuleiten, überprüft der GCKS beim Empfang einer entsprechenden Nachricht, ob es sich bei dem Absender um den Group Manager handelt (Code **??**). Der Re-Key Vorgang kann außerdem nur eingeleitet werden, wenn die ersten Schlüssel bereits verteilt wurden.

```
1 if (group && group->manager->id == client->id &&
2           group->initial_keys_distributed) {
3
4     // Schlüsselerstellung und -verteilung
5
6 }
```

Code 5.15: Überprüfung der Berechtigung zum Re-Key

Die Erzeugung neuer Schlüssel erfolgt wie bei dem oben beschriebenen Einzeleintritt, mit dem Unterschied, dass nur eine Nachricht mittels Broadcast an die Gruppe verschickt wird.

5.7 Verschlüsselung

RIOT OS bietet zum Entstehungszeitpunkt dieser Arbeit keine Möglichkeit Daten eines sich im *pktbuf* befindlichen Pakets effizient zu verschlüsseln. Die Problemstellung wird in [11] behandelt und durch die Einführung der *gnrc_pktbuf_merge()* Funktion gelöst. Dabei werden die Daten einer verketteten Liste von *snips* in ein einzelnes, neu angelegtes *snip* geschrieben. Aus dem so enstandenen Speichersegment können im Anschluss die Daten für die Verschlüsselung ausgelesen werden.

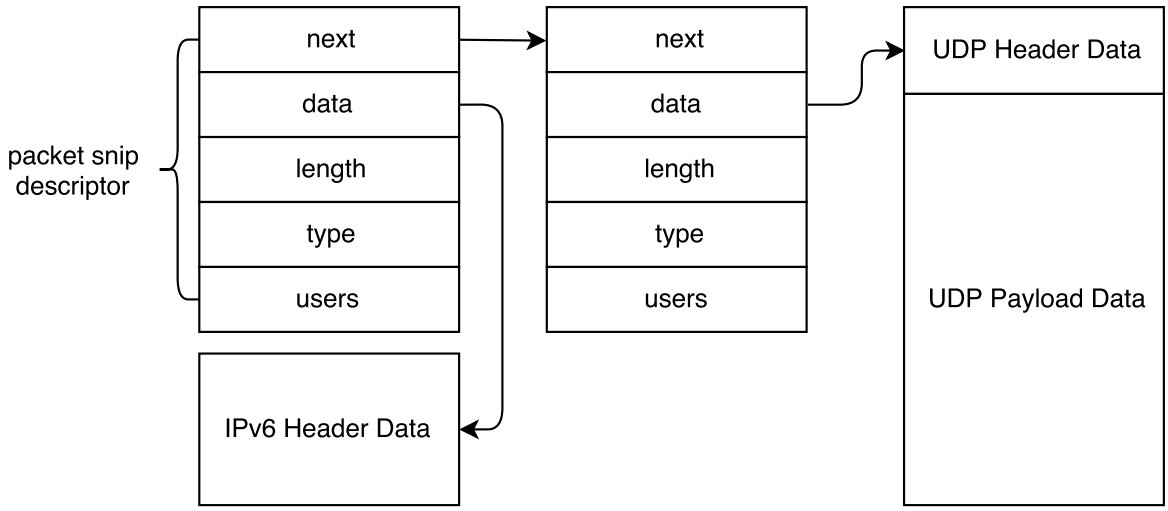

Abbildung 5.5: UDP Paket nach Anwendung der *gnrc_pktbuf_merge()* Funktion

5.8 Verwendete Module und Bibliotheken

Die folgenden Module und Bibliotheken stellen die vom Protokoll benötigten und eingesetzten kryptographischen und arithmetischen Operationen zur Verfügung.

RIOT Module Besonders nennenswert sind die zwei RIOT Module *Hashes* [24] und *Crypto* [23]. Beide lassen sich über das *Makefile* des Programms einbinden und stellen die eingesetzten kryptographischen Hashverfahren sowie Verschlüsselungsmethoden zur Verfügung.

GNU Multiple Precision Arithmetic Library [8] Für die Berechnungen des CRTs werden Zahlentypen mit größerer Genauigkeit benötigt, als von Standard C unterstützt werden. Da der Großteil der durch GMP bereitgestellten Funktionen in dieser Implementierung keine Verwendung findet, wurde stattdessen die kleinere Bibliothek *mini-gmp* benutzt. Diese stellt ebenfalls alle für die Arithmetik großer Zahlen notwendigen Funktionen bereit.

micro-ecc [22] RIOT stellt keine Funktionen für einen Diffie-Hellman-Schlüsselaustausch zur Verfügung. Die *micro-ecc* Bibliothek kann allerdings über das von RIOT zur Verfügung gestellte *package interface* eingebunden werden. *micro-ecc* stellt eine Reihe von elliptischen Kurven zur Verfügung, unter anderem *secp256r1*, welche standardmäßig in der Implementierung verwendet wird.

——— Ende des Zitats. ———

6 Evaluation

——— Der folgende Text ist ein direktes Zitat von [7]. ———

as in Kapitel 4 spezifizierte und in Kapitel 5 umgesetzte Protokoll wurde gegen die in Kapitel 3 festgelegten Anforderungen getestet.

6.1 Testumgebung und Szenario

Als Testumgebung dient der von RIOT OS zur Verfügung gestellte *native port*. Dieser ermöglicht das Ausführen von RIOT OS als Prozess in einem *NIX System, ohne dabei auf die eigentliche Hardware angewiesen zu sein. Diese ist in Abbildung 6.1 dargestellt. Auf der rechten Seite wird das Serverprogramm ausgeführt, auf der linken Seite insgesamt drei Clients. Durch die in Kapitel 6.2 beschriebenen Probleme verliefen Tests auf mobilen Endgeräten bislang nicht erfolgreich. Da die CPU-Leistung im *native port* nicht künstlich gedrosselt werden kann, ist eine qualitative Aussage über die Geschwindigkeit der Implementierung zu diesem Zeitpunkt nicht möglich. Die Ergebnisse aus [11] können als Referenzwerte hergenommen werden. Der darin entwickelte Client dient der Implementierung teilweise als Grundlage.

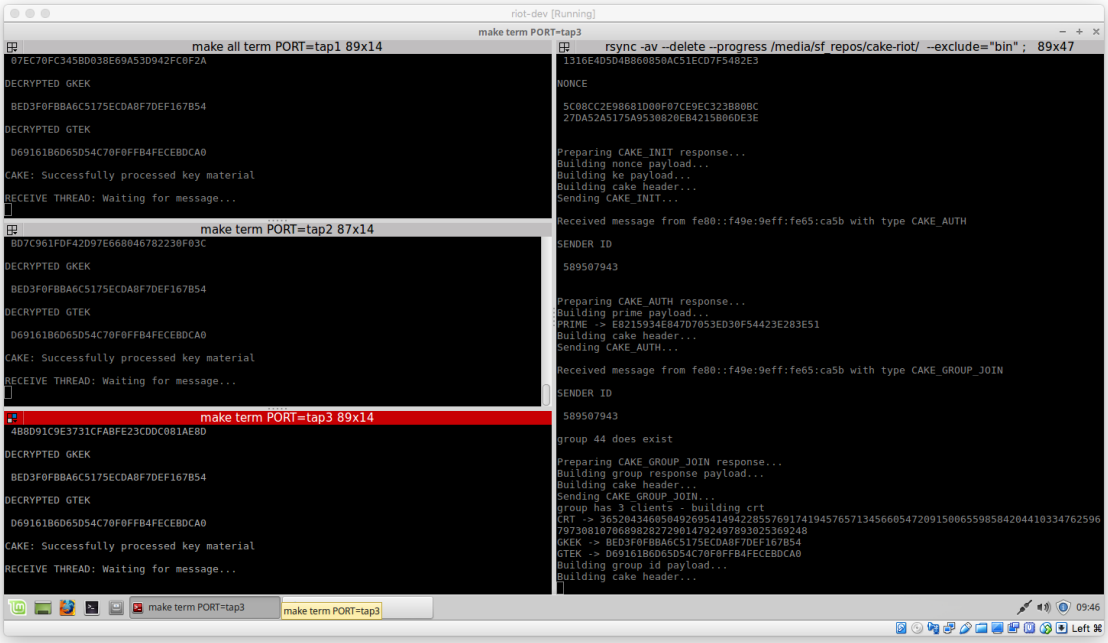

Abbildung 6.1: Testumgebung

Das in Kapitel 2.1 beschriebene Szenario wurde manuell getestet.

6.2 Bewertung

CAKE und das darum liegende Protokoll, erfüllen alle in 3.2.1 beschriebenen Anforderungen. Unautorisierte Nutzer, die in den Besitz von Nachrichten gelangen, können diese nicht entschlüsseln, da sie nicht über den dafür notwendigen GTEK verfügen. Ebensowenig können sie ein abgefangenes CRT lösen, wenn sie sich nicht unter den dafür bestimmten Empfängern befinden, da ihr privater Schlüssel und Primzahl nicht in die Berechnung mit eingeflossen ist. Diese Eigenschaft gewährleistet auch die Forward Secrecy. Verlässt ein Teilnehmer eine Gruppe oder wird er von dieser ausgeschlossen, so folgt zwangsläufig die Berechnung eines neuen CRTs, welches vom ausgeschlossenen Teilnehmer nicht gelöst werden kann. Der entstehende Rechenaufwand wird hierbei minimiert, da das CRT nicht über alle Knoten des ternären Baumes berechnet werden muss, sondern nur über einen Teilbaum, in dem sich der ehemalige Teilnehmer befand. Die Schlüsselunabhängikeit ist dadurch gewährleistet, dass bei Verschlüsselung des GTEKs mit dem GKEK eine XOR-Verknüpfung angewandt wird, welche ohne Kenntnis über den eigentlichen Schlüssel nachweislich nicht zu dechiffrieren ist [16].

Speicherbedarf

Die Datenstrukturen des System wurden weitestgehends so definiert, dass sie auf dynamische Speicherzuweisung verzichten. Bei der Berechnung des CRTs hat sich dieser Versuch als problematisch erwiesen, da der benötigte Speicher von der Anzahl der Clients abhängt, und somit erst zur Laufzeit ermittelt werden kann. Dem *mpz_t* Struct, durch welches die Primzahlen und KEKs bei der Berechnung dargestellt werden, kann zwar ein statischer Bereich im Speicher zugewiesen werden, dieser müsste aber so groß sein, dass das Produkt der Primzahlen aller n Clients einer Gruppe abgebildet werden kann. Bei einer maximalen Anzahl von 81 Clients, und einer Primzahlengröße von 32 Byte, entspräche dies im schlimmsten Fall einem Speicherbereich von mindestens $121 * 32\,\text{Byte} = 3872\,\text{Byte}$, da hier noch interne Pointer des *mpz_t* Structs dazugerechnet werden müssen.

——— Ende des Zitats. ———

7 Zusammenfassung und Ausblick

In diesem Kapitel werden Aspekte angeprochen, die zukünftige Arbeiten berücksichtigen können, um aufbauend auf das in dieser Arbeit protokollierte, auf CAKE basierende hybride Gruppenschlüsselmanagementprotokoll zu erweitern.

Tritt ein neuer Teilnehmer einer gesicherten Gruppenkommunikation bei, so sieht CAKE vor, eine Unicast an den neuen Teilnehmer zu senden und eine Multicast an die restlichen Teilnehmer der Gruppe. Der Nachrichtenaufwand aus Sicht des GCKS beträgt dabei somit zwei. Erst bei einem Masseneintritt wird die Berechnung des CRT als Alternative für das äquivalente Vorgehen beim Einzeleintritt vorgeschlagen [12] (siehe auch 4.3.2 und 4.3.3). Die Anzahl der zu versendenden Nachrichten vom GCKS könnte im Algorithmus des Einzeleintritts auf eins verringert werden, indem wie bei der initialen Erzeugung der Gruppe und der Leave Operation ein CRT berechnet wird. Dabei fließt die Primzahl und sein geheimer Schlüssel des neuen Teilnehmers in die Berechungen des CRT gemeinsam mit den Schlüsselpaaren der anderen Gruppenteilnehmer mit ein. Der entstandene Secure Lock kann anschließend per Multicast verschickt werden. Auf diese Weise reduziert sich die zu versendende Nachricht auf eins und die Backward Secrecy bleibt erhalten. Im Zusammenhang des beschriebenen Szenarios ist die Funkübertragung teuer und sollte so gering wie möglich gehalten werden. Dadurch wäre diese Methode vorteilhafter. Auf der anderen Seite muss der dadurch verbundene erhöhte Rechenaufwand berücksichtigt werden, der für den GCKS entsteht. Hierbei wird deutlich, dass der Vorteil der Nachrichtenanzahl zum Nachteil der Rechenpower des GCKS wird.

Möchte ein Gruppenteilnehmer den restlichen Gruppenteilnehmern etwas mitteilen, verschlüsselt er die Nachricht N mit dem $GTEK_{aktuell}$ und versendet diese. Kommt ein Rekey R zustande und wird dadurch der Gruppenschlüssel aktualisiert - bevor die Teilnehmer die Nachricht N empfangen - so können diese die Mitteilung nicht entschlüsseln, weil der Gruppenschlüssel aktualisiert wurde. Dies kann so gelöst werden, indem mithilfe des NTP Protokolls in jeder Nachricht ein Zeitstempel hinzugefügt wird und die Gruppenteilnehmer mindestens die letzten zwei Gruppenschlüssel abspeichern. So kann zumindest der passende Gruppenschlüssel für die Nachrichten ausgewählt werden, die vor und nach dem Rekey R erstellt und versandt wurden.

Die initiale Erzeugung einer gesicherten Gruppenkommunikation und die Beitrittsoperation benötigen keinerlei zwingenden Verwaltungsmanagement der in 4.3.3 beschriebenen Baumstruktur. Auch die in diesen zwei Gruppenoperationen notwendigen Datenpakete, welche vom GCKS zu den entsprechenden authorisierten Clients gelangen, können ohne Informationen des ternären Baumes gebildet und verschickt werden. Zwingend notwendig ist die Verwaltung des Schlüsselbaumes aber im Zusammenhang eines Austritts, da der verkleinerte CRT, der berechnet wird, die Schlüsselpaare aus KEK und Primzahl aus den entsprechenden Baumknoten benötigt. Werden die Arbeiten an der Baumstruktur nur in der Leave Opera-

tion durchgeführt, entsteht ein sehr hoher Nachrichtenaufwand bei jeder Austrittsoperation. So würde bei der ersten Leave Operation der ungünstigste Fall eintreffen. Der *ungünstigste Fall* ist, von Seiten des GCKS n Nachrichten verschicken zu müssen, wobei n die Anzahl der restlichen Gruppenteilnehmer nach einem Austritt beschreibt. Dies ist deshalb der Fall, da die Gruppenmitglieder keine Kenntnisse über ihren Pfad bis hin zur Wurzel haben. Die Arbeiten um den Schlüsselbaum könnten teilweise im Rahmen der Beitrittsoperation verlagert werden, indem den entsprechenden Teilnehmern nähere Informationen über ihren Pfad bis hin zur Wurzel mitgeteilt werden.
Zusätzlich kann im Schlüsselbaum eine Semantik entworfen werden, die den Gruppenteilnehmern ermöglicht, den zum Auflösen des CRT benötigten Knoten selbst zu errechnen, nachdem sie aufgrund eines Austritts ein neues Datenpaket erhalten haben, das in Form des aufzubrechenden Secure Locks den neuen Gruppenschlüssel enthält.

Das Ziel dieser Arbeit wurde erreicht: Die Schlüsselberechnungen auf Basis von CAKE wurden erfolgreich implementiert. Zusätzlich wurde CAKE mit IKEv2 kombiniert. Diese Arbeit dient zur Weiterentwicklung von CAKE. Erwähnenswert ist insbesondere die Nachrichtenanzahl bei der Aktualisierung des Gruppenschlüssels, die vom GCKS zum Client erfolgt. Die geringe Anzahl der Nachrichten bei den Gruppenoperationen werden durch den Algorithmus des *Chinese Remainder Theorem* möglich gemacht. CAKE befindet sich im Anfangsstadium und verdient eine ausführliche Trennung sowie eine umfassendere Erörterung des Protokoll- und Architekturdesigns, die in einem RFC protokolliert werden könnte.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Emmanuel Baccelli u. a. „RIOT: One OS to rule them all in the IoT“. Diss. INRIA, 2012.

[2] Friedrich L. Bauer. „Entzifferte Geheimnisse - Methoden und Maximen der Kryptologie“. In: Berlin Heidelberg New York: Springer-Verlag, 2013. Kap. 8.8 Individuelle Einmal-Schlüssel. ISBN: 978-3-642-58345-2.

[3] M. Baugher u. a. *The Internet Key Exchange (IKE)*. RFC 4046. Apr. 2005. URL: https://rfc-editor.org/rfc/rfc4046.txt.

[4] Guang-Huei Chiou und Wen-Tsuen Chen. „Secure broadcasting using the secure lock“. In: *IEEE Transactions on Software Engineering* 15.8 (Aug. 1989), S. 929–934. ISSN: 0098-5589. DOI: 10.1109/32.31350.

[5] A. Dunkels, B. Gronvall und T. Voigt. „Contiki - a lightweight and flexible operating system for tiny networked sensors“. In: *29th Annual IEEE International Conference on Local Computer Networks*. Nov. 2004, S. 455–462. DOI: 10.1109/LCN.2004.38.

[6] D. Eastlake, J. Schiller und S. Crocker. *Randomness Requirements for Security*. BCP 106. http://www.rfc-editor.org/rfc/rfc4086.txt. RFC Editor, Juni 2005. URL: http://www.rfc-editor.org/rfc/rfc4086.txt.

[7] Edgar Goetzendorff. „CAKE - Hybrides Gruppenschlüsselmanagementprotokoll für RIOT OS“. Bachelor's Thesis. Ludiwg-Maximilians-Universität München, Apr. 2018.

[8] Torbjörn Granlund. *GNU MP*. Version 6.0.0. Free Software Foundation. 2014.

[9] T. Hardjono, S. Rowles und B. Weis. *The Group Domain of Interpretation*. RFC 6407. Okt. 2011. URL: https://rfc-editor.org/rfc/rfc6407.txt.

[10] D. Harkins und D. Carrel. *The Internet Key Exchange (IKE)*. RFC 2409. Nov. 1998. URL: https://rfc-editor.org/rfc/rfc2409.txt.

[11] Tobias Heider. „Minimal G-IKEv2 implementation for RIOT OS“. Bachelor's Thesis. Ludiwg-Maximilians-Universität München, Apr. 2017.

[12] Peter Hillmann, Marcus Knüpfer und Gabi Dreo Rodosek. „CAKE: Hybrides Gruppen-Schlüssel-Management Verfahren“. In: *10. DFN-Forum Kommunikationstechnologien*. Hrsg. von Paul Müller u. a. Bonn: Gesellschaft für Informatik e.V., 2017, S. 31–40.

[13] C. Kaufman u. a. *Internet Key Exchange Protocol Version 2 (IKEv2)*. STD 79. http://www.rfc-editor.org/rfc/rfc7296.txt. RFC Editor, Okt. 2014. URL: http://www.rfc-editor.org/rfc/rfc7296.txt.

[14] Martine Lenders. „Analysis and Comparison of Embedded Network Stacks“. Master's Thesis. Freie Universität Berlin, Apr. 2016.

[15] P. A. Levis. „TinyOS: An Open Operating System for Wireless Sensor Networks (Invited Seminar)“. In: *7th International Conference on Mobile Data Management (MDM'06)*. Mai 2006, S. 63–63. DOI: 10.1109/MDM.2006.151.

[16] C. Matt und U. Maurer. „The one-time pad revisited“. In: *2013 IEEE International Symposium on Information Theory*. Juli 2013, S. 2706–2710. DOI: 10.1109/ISIT.2013.6620718.

[17] D. Maughan u. a. *Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 2408. Nov. 1998. URL: https://rfc-editor.org/rfc/rfc2408.txt.

[18] Sandro Rafaeli und David Hutchison. „A Survey of Key Management for Secure Group Communication“. In: 35 (Sep. 2003), S. 309–329.

[19] Bundesamt für Sicherheit in der Informationstechnik. „BSI-Grundschutz Katalog“. In: 1996. Kap. M 2.46 Geeignetes Schlüsselmanagement. URL: http://www.bsi.de/gshb/deutsch/index.htm.

[20] Brian Weis, Yoav Nir und Valery Smyslov. *Group Key Management using IKEv2*. Internet-Draft draft-yeung-g-ikev2-13. http://www.ietf.org/internet-drafts/draft-yeung-g-ikev2-13.txt. IETF Secretariat, März 2018. URL: http://www.ietf.org/internet-drafts/draft-yeung-g-ikev2-13.txt.

[21] H. Will, K. Schleiser und J. Schiller. „A real-time kernel for wireless sensor networks employed in rescue scenarios“. In: *2009 IEEE 34th Conference on Local Computer Networks*. Okt. 2009, S. 834–841. DOI: 10.1109/LCN.2009.5355049.

Web-Literatur

[22] *micro-ecc Webseite*. URL: http://kmackay.ca/micro-ecc/ (besucht am 05. 04. 2018).

[23] *RIOT Crypto Modul*. URL: http://riot-os.org/api/group__sys__crypto.html (besucht am 05. 04. 2018).

[24] *RIOT Hashes Module*. URL: http://riot-os.org/api/group__sys__hashes.html (besucht am 05. 04. 2018).